fМИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» им.В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Пояснительная записка к курсовой работе
по дисциплине «Объектно-ориентированное программирование»

Выполнил студент гр. 9308 _____ Яловега Н. В.

Проверил _____ Гречухин М. Н.

Санкт-Петербург

2021

Оглавление

Техническое задание

Введение

Разработать ПК (программный комплекс) для директора завода по производству металлических изделий. В ПК должна храниться информация о рабочих с указанием специализации, менеджерах, клиентах и договорах. Директор может добавлять, изменять и удалять информацию.

Цель и задачи

Целью курсового проектирования является закрепление и углубление теоретических знаний, приобретение практических навыков по проектированию и разработке программного обеспечения на объектно-ориентированном языке Java.

В задачи курсового проектирования входят:

- изучение особенностей конкретной предметной области, относящейся к заданию на курсовой проект, и разработка технического задания на программный комплекс (ПК);
- объектно-ориентированное проектирование ПК с использованием языка UML;
- разработка ПК на объектно-ориентированном языке;
- написание программной документации.

Основание для разработки

Основанием для разработки ПК – это курсовой проект по дисциплине «Объектно-ориентированное программирование».

Требования к программе

Требования к функциональным характеристикам

ПК должен обеспечивать выполнение следующих функций:

- Просмотр, добавление, изменение базы данных (БД);
- Выдача справочной информации, хранимой в БД.
- Список всех договоров указанного клиента
- Список всех клиентов указанного менеджера
- Отчёт об участии выбранного работника в выполнении договоров за указанный период (даты с и по)
- Перечень всех договоров, заключенных за указанный период (даты с и по)
- Отчёт о работе завода за выбранный месяц: сколько договоров было заключено, сколько ранее заключенных договоров выполнено, на какую сумму
- Отчёт о просроченных договорах

Требования к организации и форме представления выходных данных

Выходные данные должны быть представлены в виде таблицы содержащей запрос пользователя.

Требования к организации и форме представления входных данных

Ввод исходных данных должен осуществляется в режиме диалога. Вводимые данные являются значениями характеристик редактируемых/добавляемых объектов.

Требования к надёжности

ПК должен устойчиво функционировать при соблюдении гарантии устойчивого функционирования ОС и СУБД. Должен быть обеспечен контроль входных данных на предмет соответствия предполагаемому типу.

Условия эксплуатации

Выполнение ПК своих функций должным образом будет соблюдаться при наличии стабильного подключения к СУБД.

Требование к информационной и программной совместимости

Выходная и входная информация ПК должна быть удобны для работы.

ПК должен быть выполнен на языке программирования Java и должен быть совместим с операционными системами GNU/Linux и Windows.

Обязательными требованиями при разработке кода ПК являются использование следующих конструкций языка Java:

- закрытые и открытые члены классов;
- наследование;
- конструкторы с параметрами;
- виртуальные функции;
- обработка исключительных ситуаций;
- динамическое создание объектов.

Требования к программной документации

Документация должна быть представлена в следующем составе:

- описание процесса проектирования ПК;
- руководство оператора;
- исходные тексты ПК.

Проектирование ПК

Описание вариантов использования ПК

Описание функциональных требований осуществляется на этапе проектирования комплекса. Для того чтобы детализировать требования, необходимо выделить процессы, происходящие в заданной предметной области. Описание таких процессов на UML выполняется в виде прецедентов (use case). Диаграмма прецедентов представлена на рисунке 1.

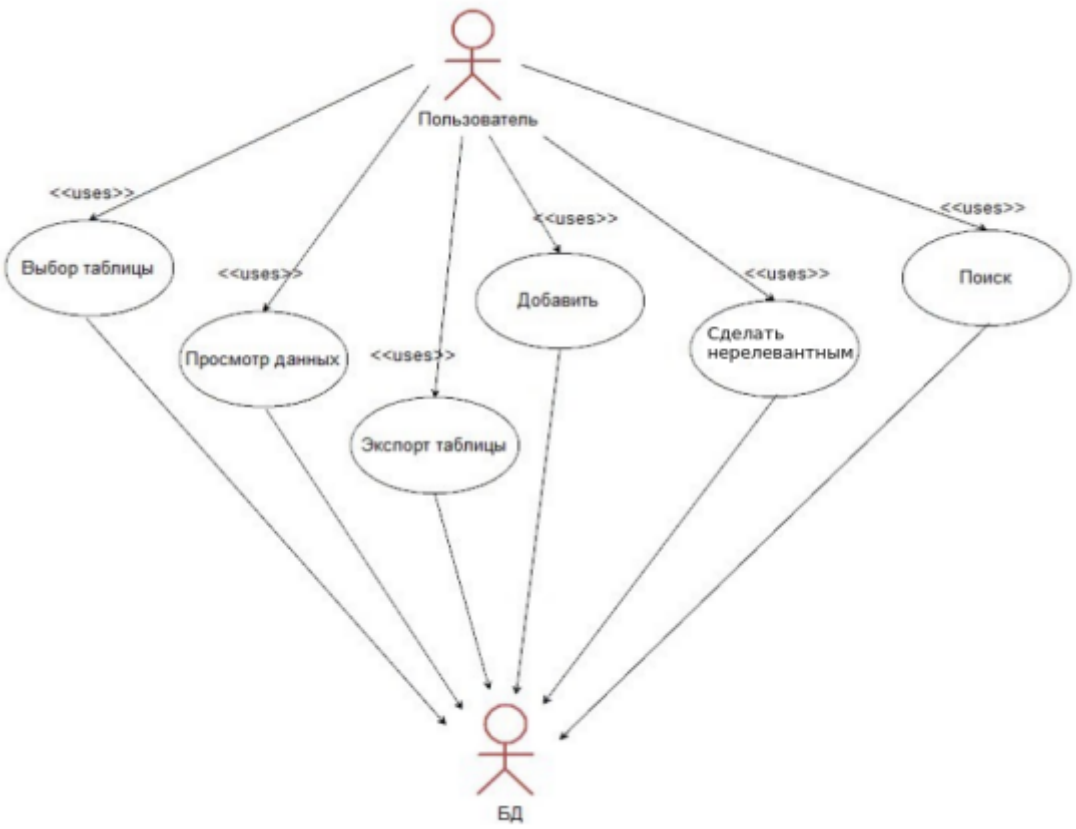


Рисунок 1: Диаграмма прецедентов

Создание прототипа интерфейса пользователя

Сначала необходимо описать экранные формы и элементы управления (кнопки, выпадающие списки, чекбоксы и др.). Такое описание представлено в таблице 1.

Таблица 1. Описание графических форм

Экранная форма	**Элементы управления**
Главное меню	Кнопка «Список доступных профессий» Кнопка «Список рабоников завода» Кнопка «Список менеджеров завода» Кнопка «Список клиентов завода» Кнопка «Список договоров завода» Кнопка «Открыть справку о программе»
Таблица сотрудников/менеджеров/клиентов/контрактов/специализаций	Поле поиска Кнопка «Добавление записи» Кнопка «Редактирования записи» Кнопка «Удаления записи» Кнопка «Создания отчета» Кнопка «Поиск»
Добавление сотрудников/менеджеров/клиентов/контрактов/специализаций	Поля для заполнения сведений Кнопка «Добавить»
Редактирование сотрудников/менеджеров/клиентов/контрактов/специализаций	Поля для редактирования сведений Кнопка «Добавить»
Справка о программе	

Примеры спроектированного пользовательского интерфейса представлены на рисунках 2, 3, 4, 5, 6.

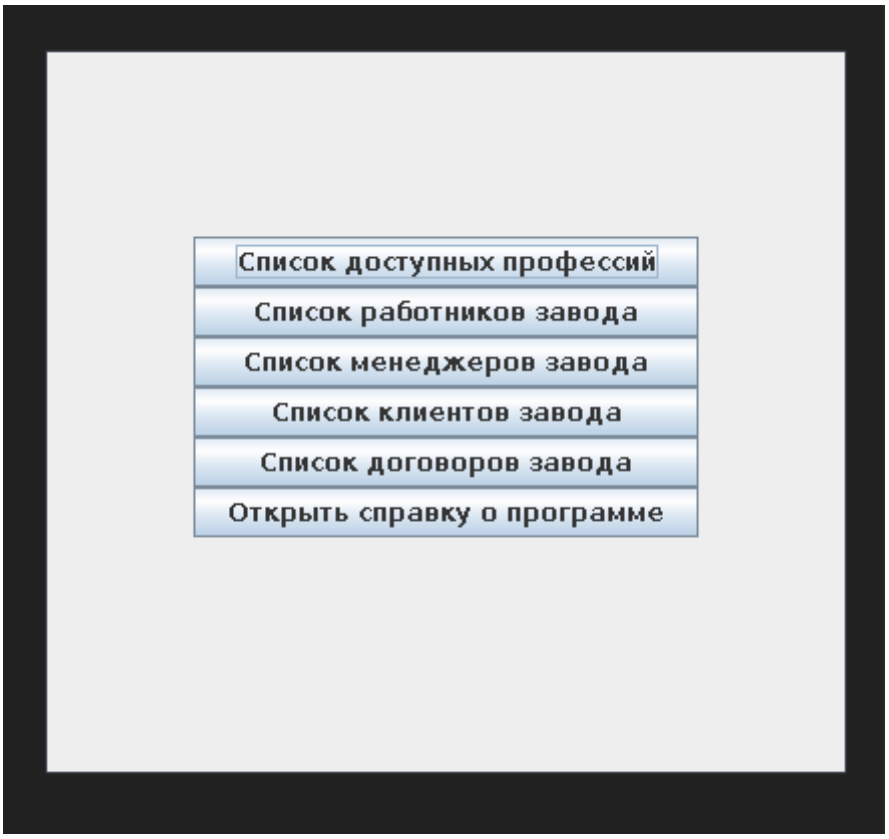
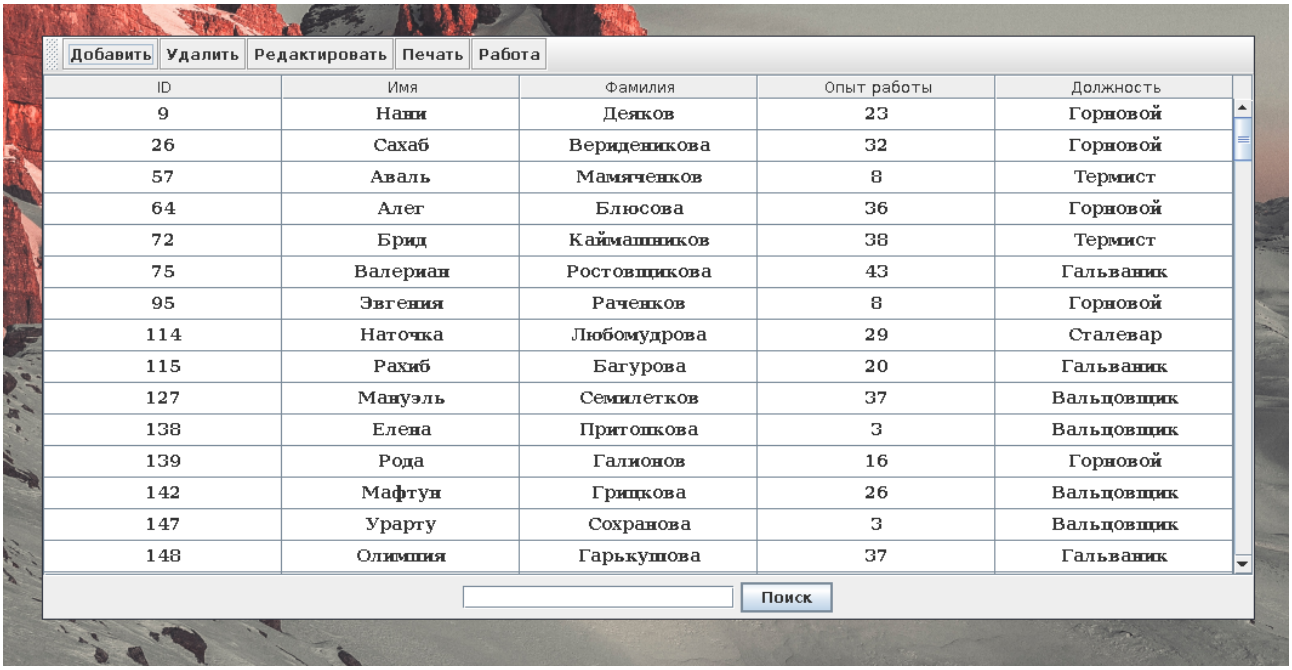


Рисунок 2: Главное меню



ID	Имя	Фамилия	Опыт работы	Должность
9	Нани	Деяков	23	Горновой
26	Сахаб	Вериденикова	32	Горновой
57	Аваль	Мамяченков	8	Термист
64	Алег	Блюсова	36	Горновой
72	Брид	Каймашников	38	Термист
75	Валериан	Ростовщикова	43	Гальваник
95	Эвгения	Раченков	8	Горновой
114	Наточка	Любомудрова	29	Сталевар
115	Рахиб	Багурова	20	Гальваник
127	Мануэль	Семилетков	37	Вальцовщик
138	Елена	Притошкова	3	Вальцовщик
139	Рода	Галионов	16	Горновой
142	Мафтун	Грицкова	26	Вальцовщик
147	Урарту	Сохранова	3	Вальцовщик
148	Олимпия	Гарькушова	37	Гальваник

Рисунок 3: Список работников завода

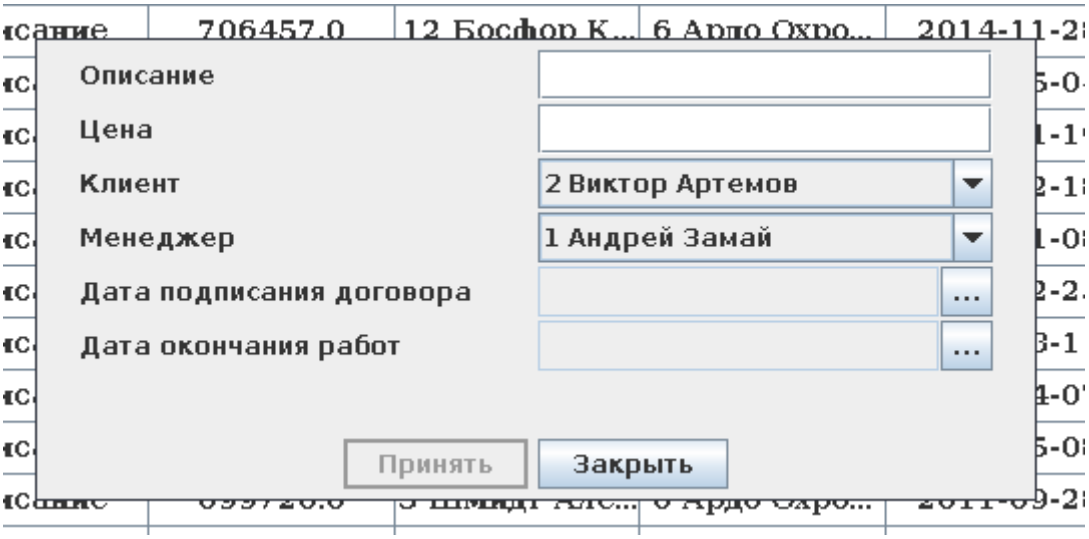


Рисунок 4: Диалог добавления договора

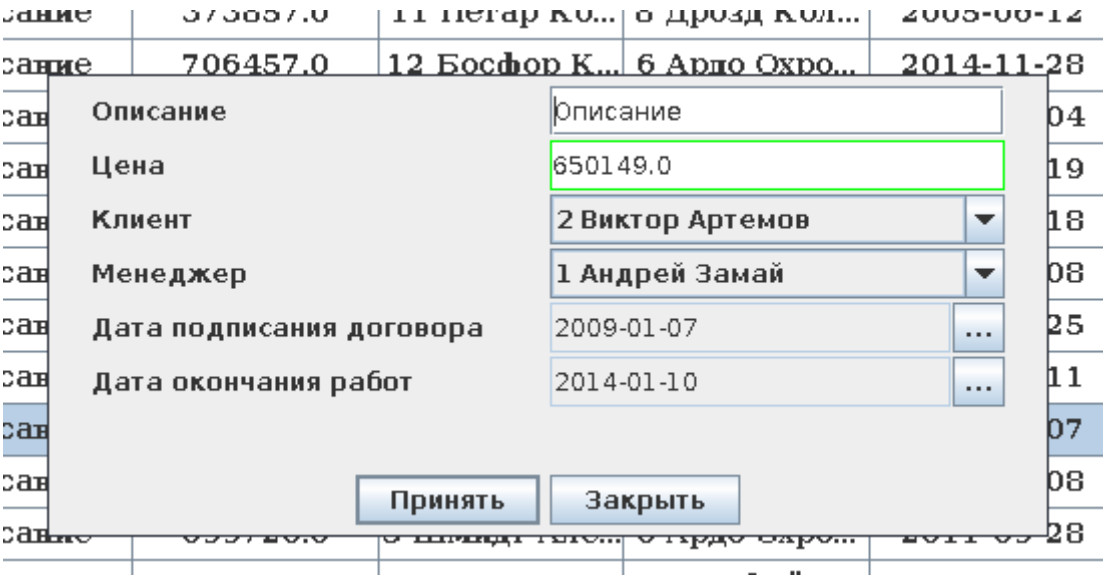


Рисунок 5: Диалог изменения договора

Рисунок 6: Справка о программе

Построение диаграммы программных классов

Диаграмма классов (class diagram) строится на основе объектной модели. В описание класса указываются три раздела: имя класса, состав компонентов класса и методы класса. Графически класс изображается в виде прямоугольника. Такая диаграмма представлена на рисунке 7.

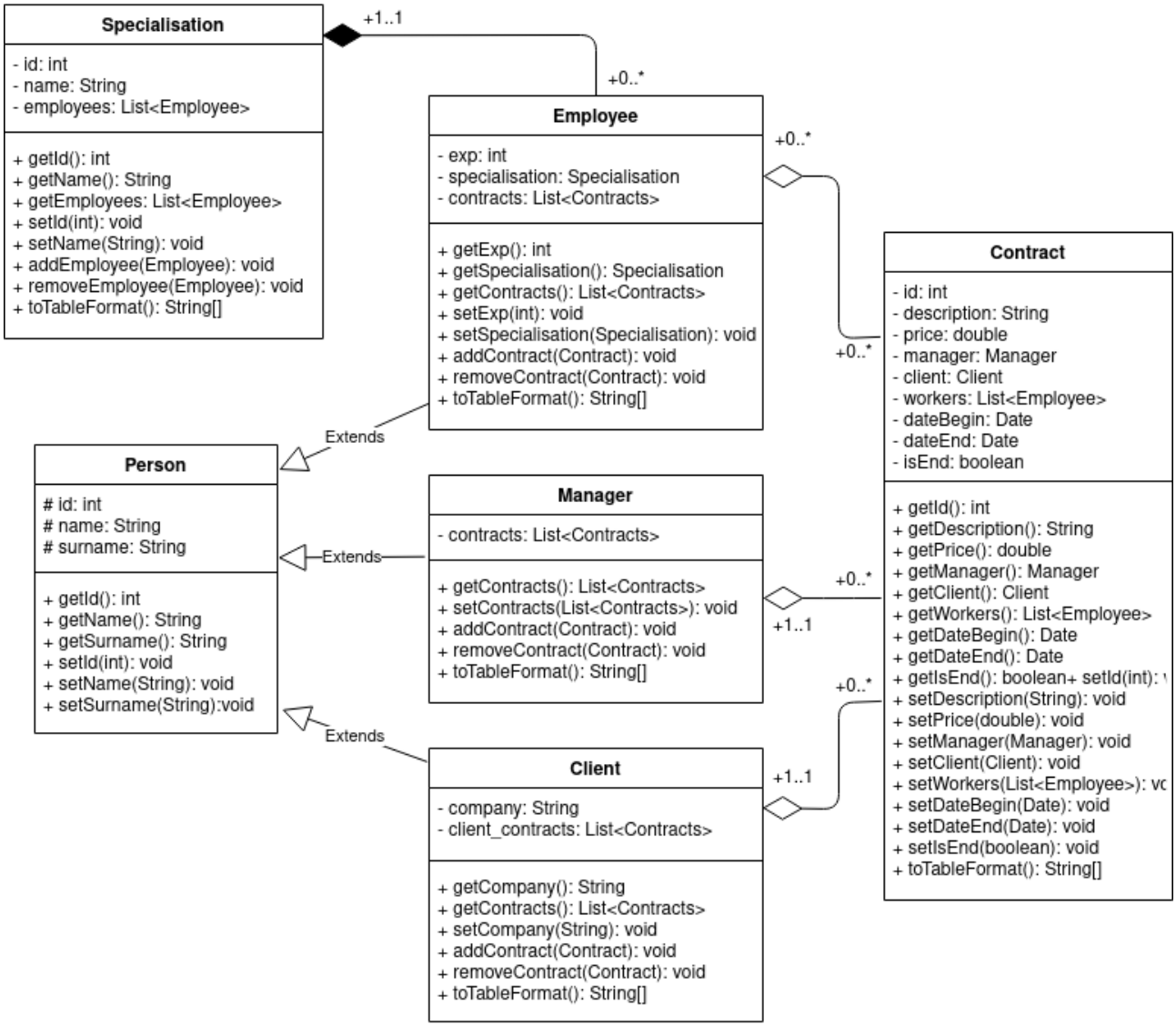


Рисунок 7: UML-диаграмма классов

Описание классов

С помощью утилиты javadoc была сформирована документация. Примеры такой документации представлены на рисунках 8, 9 . Команда для генерации такой документации в интерпретаторе bash: «javadoc $(find /project/dir -name "*.java")». Архив с javadoc приложен к проекту.

Class Person

java.lang.Object
    Factory.model.Person

Direct Known Subclasses:

Client, Employee, Manager

```
@MappedSuperclass
public class Person
extends java.lang.Object
```

Класс человека

Version:

0.1

Author:

Яловега Никита 9308

Field Summary

Fields

Modifier and Type	Field	Description
protected int	id	Поле уникального идентификатора
protected java.lang.String	name	Поле имени человека
protected java.lang.String	surname	Поле фимилии человека

Рисунок 8: Пример javadoc 1

Constructor Summary

Constructors

Constructor	Description
Person()	Стандартный контруктор человека
Person(java.lang.String name, java.lang.String surname)	Конструктор - создание нового объекта Person

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method	Description
int	getId()	Функция получения значения поля id
java.lang.String	getName()	Функция получения значения поля name
java.lang.String	getSurname()	Функция получения значения поля surname
void	setId(int newID)	Функция определения значения поля id
void	setName(java.lang.String newName)	Процедура определения значения поля name
void	setSurname(java.lang.String newSurname)	Процедура определения значения поля name

Рисунок 9: Пример javadoc 2

App.java

Главный класс, запускающий приложение.

Описание полей класса представлено в таблице 2.

Таблица 2 - описание полей класса App

Название	Тип данных	Семантика
private static final log	Logger	Логгер класса

Описание методов класса представлено в таблице 3.

Таблица 3 - описание методов класса App

Название метода	Возвращаемый тип	Описание параметров	Назначение
public static main	void	String[] args — аргументы программе	Главный метод main

Person.java

Класс человек. От него наследуются Employee, Manager, Client.

Описание полей класса представлено в таблице 4.

Таблица 4 - описание полей класса Person

Название	Тип данных	Семантика
private id	int	Уникальный идентификатор
private name	String	Имя человека
private surname	String	Фамилия человека

Описание методов класса представлено в таблице 5.

Таблица 5 - описание методов класса Person

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getId	int		Получить ID человека
public getName	String		Получить имя человека
public getSurname	String		Получить фамилию человека
public setId	void	int newId	Установить ID человека
public setName	void	String newName – имя	Установить имя человека
public setSurname	void	String newSurname – фамилия	Установить фамилию

Client.java

Класс, представляющий клиента, наследник класса Person

Описание полей класса представлено в таблице 6.

Таблица 6 - описание полей класса Client

Название	Тип данных	Семантика
private company	String	Название компании клиента
private client_contacts	List<Contract>	Договоры клиента

Описание методов класса представлено в таблице 7.

Таблица 7 - описание методов класса Client

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getCompany	String		Получения значения поля company
public getContracts	List<Contract>		Получение значения поля client_contacts
public setCompany	void	String newCompany - компания	Установить название компании клиента
public addContract	void	Contract newContract — новый договор	Добавление договоров клиенту
public removeContract	void	Contact c — договор	Удаление договора у клиента
public toTableFormat	String[]		Получение всей информации об объекте

Employee.java

Класс, представляющий сотрудника завода. наследник класса Person

Описание полей класса представлено в таблице 8.

Таблица 8 - описание полей класса Employee

Название	Тип данных	Семантика
private contracts	List<Contract>	Контракты, в которых участвует рабочий
private exp	int	Опыт работы
private specialisation	Specialisation	Поле профессии рабочего

Описание методов класса представлено в таблице 9.

Таблица 9 - описание методов класса Employee

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getExp	int		Получения значения поля exp
public getContracts	List<Contract>		Получение значения поля contacts
public getSpecialisation	Specialisation		Получение значения поля specialisation
public addContract	void	Contract newContract — новый договор	Добавление договоров рабочему
public removeContract	void	Contact c — договор	Удаление договора у рабочего
public setExp	void	int newexp — опыт работы	Установить опыт работы
public setSpecialisation	void	Specialisation d — профессия	Установить профессию
public toTableFormat	String[]		Получение всей информации об объекте

Manager.java

Класс, представляющий менеджера завода. наследник класса Person

Описание полей класса представлено в таблице 10.

Таблица 10 - описание полей класса Manager

Название	Тип данных	Семантика
private contracts	List<Contract>	Контракты, которые подписывал менеджер

Описание методов класса представлено в таблице 11.

Таблица 11 - описание методов класса Manager

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getContracts	List<Contract>		Получение значения поля contacts
public addContract	void	Contract newContract — новый договор	Добавление договоров менеджеру
public removeContract	void	Contact c — договор	Удаление договора у менеджера
public toTableFormat	String[]		Получение всей информации об объекте

Specialisation.java

Класс специализации рабочего

Описание полей класса представлено в таблице 12.

Таблица 12 - описание полей класса Specialisation

Название	Тип данных	Семантика
private id	int	Уникальный идентификтор профессии
private employees	List<Employee>	Рабочие данной професии
private name	String	Название профессии

Описание методов класса представлено в таблице 13.

Таблица 13 - описание методов класса Specialisation

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getId	int		Получения значения поля id
public getEmployees	List<Employee>		Получение значения поля employees
public getName	String		Получение значения поля name
public addEmployee	void	Employee newEmployee— новый рабочий	Добавление нового рабочего
public removeEmployee	void	Employee e — рабочий	Удаление рабочего
public setId	void	int newID— новый идентификатор	Установить id
public setName	void	String newName— название	Установить название профессии
public toTableFormat	String[]		Получение всей информации об объекте

Contract.java

Класс контракта

Описание полей класса представлено в таблице 14.

Таблица 14 - описание полей класса Contract

Название	Тип данных	Семантика
private id	int	Уникальный идентификтор контракта
private price	double	Цена контракта
private description	String	Описание условий контракта
private workers	List<Employee>	Рабочие, выполняющие условия контракта
private manager	Manager	Менеджер, подписавший контракт
private client	Client	Клиент, подписавший контракт
private dateBegin	Date	Дата начала действия контракта
private dateEnd	Date	Дата окончания действия контракта
private isEnd	boolean	Завершили ли выполнение контракта

Описание методов класса представлено в таблице 15.

Таблица 15 - описание методов класса Contract

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getId	int		Получения значения поля id
public getPrice	int		Получения значения поля price
public getDescription	String		Получение значения поля description
public getWorkers	List<Employee>		Получение значения поля workers
public getManager	Manager		Получение значения поля manager
public getClient	Client		Получение значения поля client
public getDateBegin	Date		Получение значения поля dateBegin

public getDateEnd	Date		Получение значения поля dateEnd
public getIsEnd	boolean		Получение значения поля isEnd
public addWorker	void	Employee newEmployee— новый рабочий	Добавление нового рабочего
public removeWorker	void	Employee e — рабочий	Удаление рабочего
public setId	void	int newID— идентификатор	Установить id
public setPrice	void	int newPrice— цена	Установить цену
public setDescription	void	String newDescription— название	Установить описание
public setManager	void	Manager newManager— менеджер	Установить менеджера
public setClient	void	Client newClient— клиент	Установить клиента
public setDateBegin	void	Date newDate — дата начала	Установить дату начала
public setDateEnd	void	Date newDate — дата окончания	Установить дату конца
public setIsEnd	void	boolean i — новое состояние	Установить состояние выполнения
public toTableFormat	String[]		Получение всей информации об объекте

EmployeeDAO.java

Data Access Object для класса Employee

Описание полей класса представлено в таблице 16.

Таблица 16 - описание полей класса EmployeeDAO

Название	Тип данных	Семантика
private currentSession	Session	Текущая сессия
private currentTransaction	Transaction	Текущая транзакция

Описание методов класса представлено в таблице 17.

Таблица 17 - описание методов класса EmployeeDAO

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getCurrentSession	Session		Получить текущую сессию
public getCurrentTransaction	Transaction		Получить текущую транзакцию
public openCurrentSession	Session		Открыть текущую сессию
public openCurrentSessionwithTransaction	Session		Открыть текущую сессию с транзакцией
public setCurrentSession	void	Session currentSession — сессия	Установить текущую сессию
public setCurrentTransaction	void	Transaction currentTransaction — транзакция	Установить текущую транзакцию
public closeCurrentSession	void		Закрыть текущую сессию
public closeCurrentSessionwithTransaction	void		Закрыть текущую сессию с транзакцией
public persist	void	Employee entity — рабочий	Добавить рабочего в БД
public update	void	Employee entity - рабочий	Обновить данные рабочего в БД
public delete	void	Employee entity - рабочий	Удалить рабочего из БД
public deleteAll	void		Удалить всех рабочих из БД
public findById	Employee	int id — уникальный	Найти рабочего по ID

		идентификатор	
public findAll	List<Employee>		Найти всех рабочих

ManagerDAO.java

Data Access Object для класса Manager

Описание полей класса представлено в таблице 18.

Таблица 18 - описание полей класса ManagerDAO

Название	Тип данных	Семантика
private currentSession	Session	Текущая сессия
private currentTransaction	Transaction	Текущая транзакция

Описание методов класса представлено в таблице 19.

Таблица 19 - описание методов класса ManagerDAO

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getCurrentSession	Session		Получить текущую сессию
public getCurrentTransaction	Transaction		Получить текущую транзакцию
public openCurrentSession	Session		Открыть текущую сессию
public openCurrentSessionwithTransaction	Session		Открыть текущую сессию с транзакцией
public setCurrentSession	void	Session currentSession — сессия	Установить текущую сессию
public setCurrentTransaction	void	Transaction currentTransaction — транзакция	Установить текущую транзакцию
public closeCurrentSession	void		Закрыть текущую сессию
public closeCurrentSessionwithTransaction	void		Закрыть текущую сессию с транзакцией
public persist	void	Manager entity — менеджер	Добавить рабочего в БД
public update	void	Manager entity — менеджер	Обновить данные менеджера в БД
public delete	void	Manager entity — менеджер	Удалить менеджера из БД
public deleteAll	void		Удалить всех менеджеров из БД
public findById	Manager	int id — уникальный	Найти менеджера по

		идентификатор	ID
public findAll	List<Manager>		Найти всех менеджеров

SpecialisationDAO.java

Data Access Object для класса Specialisation

Описание полей класса представлено в таблице 20.

Таблица 20 - описание полей класса SpecialisationDAO

Название	Тип данных	Семантика
private currentSession	Session	Текущая сессия
private currentTransaction	Transaction	Текущая транзакция

Описание методов класса представлено в таблице 21.

Таблица 21 - описание методов класса SpecialisationDAO

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getCurrentSession	Session		Получить текущую сессию
public getCurrentTransaction	Transaction		Получить текущую транзакцию
public openCurrentSession	Session		Открыть текущую сессию
public openCurrentSessionwithTransaction	Session		Открыть текущую сессию с транзакцией
public setCurrentSession	void	Session currentSession — сессия	Установить текущую сессию
public setCurrentTransaction	void	Transaction currentTransaction — транзакция	Установить текущую транзакцию
public closeCurrentSession	void		Закрыть текущую сессию
public closeCurrentSessionwithTransaction	void		Закрыть текущую сессию с транзакцией
public persist	void	Specialisation entity — профессия	Добавить профессию в БД
public update	void	Specialisation entity — профессия	Обновить данные профессии в БД
public delete	void	Specialisation entity — профессия	Удалить профессию из БД
public deleteAll	void		Удалить все профессии из БД
public findById	Specialisation	int id — уникальный	Найти профессии по

		идентификатор	ID
public findAll	List<Specialisation>		Найти все профессии

ContractDAO.java

Data Access Object для класса Contract

Описание полей класса представлено в таблице 22.

Таблица 22 - описание полей класса ContractDAO

Название	Тип данных	Семантика
private currentSession	Session	Текущая сессия
private currentTransaction	Transaction	Текущая транзакция

Описание методов класса представлено в таблице 23.

Таблица 23 - описание методов класса ContractDAO

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getCurrentSession	Session		Получить текущую сессию
public getCurrentTransaction	Transaction		Получить текущую транзакцию
public openCurrentSession	Session		Открыть текущую сессию
public openCurrentSessionwithTransaction	Session		Открыть текущую сессию с транзакцией
public setCurrentSession	void	Session currentSession — сессия	Установить текущую сессию
public setCurrentTransaction	void	Transaction currentTransaction — транзакция	Установить текущую транзакцию
public closeCurrentSession	void		Закрыть текущую сессию
public closeCurrentSessionwithTransaction	void		Закрыть текущую сессию с транзакцией
public persist	void	Contract entity — контракт	Добавить контракт в БД
public update	void	Contract entity — контракт	Обновить данные контракта в БД
public delete	void	Contract entity — контракт	Удалить контракт из БД
public deleteAll	void		Удалить все контракты из БД
public findById	Contract	int id — уникальный	Найти контракты по

		идентификатор	ID
public findAll	List<Contract>		Найти все контракты

ClientDAO.java

Data Access Object для класса Client

Описание полей класса представлено в таблице 24.

Таблица 24 - описание полей класса ClientDAO

Название	Тип данных	Семантика
private currentSession	Session	Текущая сессия
private currentTransaction	Transaction	Текущая транзакция

Описание методов класса представлено в таблице 25.

Таблица 25 - описание методов класса ClientDAO

Название метода	Возвращаемый тип	Описание параметров	Назначение
public getCurrentSession	Session		Получить текущую сессию
public getCurrentTransaction	Transaction		Получить текущую транзакцию
public openCurrentSession	Session		Открыть текущую сессию
public openCurrentSessionwithTransaction	Session		Открыть текущую сессию с транзакцией
public setCurrentSession	void	Session currentSession — сессия	Установить текущую сессию
public setCurrentTransaction	void	Transaction currentTransaction — транзакция	Установить текущую транзакцию
public closeCurrentSession	void		Закрыть текущую сессию
public closeCurrentSessionwithTransaction	void		Закрыть текущую сессию с транзакцией
public persist	void	Client entity — клиент	Добавить профессию в БД
public update	void	Client entity — клиент	Обновить данные клиента в БД
public delete	void	Client entity — клиент	Удалить клиента из БД
public deleteAll	void		Удалить всех клиентов из БД
public findById	Client	int id — уникальный	Найти клиента по ID

		идентификатор	
public findAll	List<Client>		Найти всех клиентов

EmployeeService.java

Service для класса Employee

Описание полей класса представлено в таблице 26.

Таблица 26 - описание полей класса EmployeeService

Название	Тип данных	Семантика
private static employeeDao	EmployeeDAO	DAO employee

Описание методов класса представлено в таблице 27.

Таблица 27 - описание методов класса EmployeeService

Название метода	Возвращаемый тип	Описание параметров	Назначение
public persist	void	Employee entity — рабочий	Добавить рабочего в БД
public update	void	Employee entity - рабочий	Обновить данные рабочего в БД
public delete	void	Employee entity - рабочий	Удалить рабочего из БД
public deleteAll	void		Удалить всех рабочих из БД
public findById	Employee	int id — уникальный идентификатор	Найти рабочего по ID
public findAll	List<Employee>		Найти всех рабочих

ManagerService.java

Service для класса Manager

Описание полей класса представлено в таблице 28.

Таблица 28 - описание полей класса ManagerService

Название	Тип данных	Семантика
private static managerDao	ManagerDAO	DAO manager

Описание методов класса представлено в таблице 29.

Таблица 29 - описание методов класса ManagerService

Название метода	Возвращаемый тип	Описание параметров	Назначение
public persist	void	Manager entity — менеджер	Добавить рабочего в БД
public update	void	Manager entity — менеджер	Обновить данные менеджера в БД
public delete	void	Manager entity — менеджер	Удалить менеджера из БД
public deleteAll	void		Удалить всех менеджеров из БД
public findById	Manager	int id — уникальный идентификатор	Найти менеджера по ID
public findAll	List<Manager>		Найти всех менеджеров

SpecialisationService.java

Service для класса Specialisation

Описание полей класса представлено в таблице 30.

Таблица 30 - описание полей класса SpecialisationService

Название	Тип данных	Семантика
private static specialosaionDao	SpecialisationDAO	DAO specialisation

Описание методов класса представлено в таблице 31.

Таблица 31 - описание методов класса SpecialisationService

Название метода	Возвращаемый тип	Описание параметров	Назначение
public persist	void	Specialisation entity — профессия	Добавить профессию в БД
public update	void	Specialisation entity — профессия	Обновить данные профессии в БД
public delete	void	Specialisation entity — профессия	Удалить профессию из БД
public deleteAll	void		Удалить все профессии из БД
public findById	Specialisation	int id — уникальный идентификатор	Найти профессии по ID
public findAll	List<Specialisation>		Найти все профессии

ContractService.java

Service для класса Contract

Описание полей класса представлено в таблице 32.

Таблица 32 - описание полей класса ContractService

Название	Тип данных	Семантика
private static ContractDao	ContractDAO	DAO contract

Описание методов класса представлено в таблице 33.

Таблица 33 - описание методов класса ContractService

Название метода	Возвращаемый тип	Описание параметров	Назначение
public persist	void	Contract entity — контракт	Добавить контракт в БД
public update	void	Contract entity — контракт	Обновить данные контракта в БД
public delete	void	Contract entity — контракт	Удалить контракт из БД
public deleteAll	void		Удалить все контракты из БД
public findById	Contract	int id — уникальный идентификатор	Найти контракты по ID
public findAll	List<Contract>		Найти все контракты

ClientService.java

Service для класса Client

Описание полей класса представлено в таблице 34.

Таблица 34 - описание полей класса ClientService

Название	Тип данных	Семантика
private static ClientDao	ClientDAO	DAO client

Описание методов класса представлено в таблице 35.

Таблица 35 - описание методов класса ClientService

Название метода	Возвращаемый тип	Описание параметров	Назначение
public persist	void	Client entity — клиент	Добавить профессию в БД
public update	void	Client entity — клиент	Обновить данные клиента в БД
public delete	void	Client entity — клиент	Удалить клиента из БД
public deleteAll	void		Удалить всех клиентов из БД
public findById	Client	int id — уникальный идентификатор	Найти клиента по ID
public findAll	List<Client>		Найти всех клиентов

HibernateSessionFactoryUtil.java

Класс организующий работу с hibernate.

Описание полей класса представлено в таблице 36.

Таблица 36- описание полей класса HibernateSeesionFactoryUtil

Название	Тип данных	Семантика
private static factory	SessionFactory	Объект, генерирующий сессию

Описание методов класса представлено в таблице 37.

Таблица 37 - описание методов класса HibernateSessionFactoryUtil

Название метода	Возвращаемый тип	Описание параметров	Назначение
public static getSessionFactory	SessionFactory		Получить объект, генерирующий сессию

ReportUtil.java

Класс, отвечающий за генерацию отчётов PDF.

Описание методов класса представлено в таблице 38.

Таблица 38 - описание методов класса ReportUtil

Название метода	Возвращаемый тип	Описание параметров	Назначение
public static print	void	String datasource - Имя файла XML с данными String xpath - Директория до полей с данными String template - Имя файла шаблона .jrxml String resultpath - Имя файла, в который будет помещен отчет	Метод генерации отчетов в формате PDF

Menu.java

Класс главного меню

Описание методов класса представлено в таблице 39.

Таблица 39 - описание методов класса Menu

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод показа главного меню

SpecialisationWindow.java

Класс приложения, визуализирующий экранную форму с профессиями

Описание полей класса представлено в таблице 40.

Таблица 40 - описание полей класса SpecialisationWindow

Название	Тип данных	Семантика
private window	JFrame	Окно приложения
private model	DefaultTableModel	Модель таблицы
private add	JButton	Добавить
private delete	JButton	Удалить
private edit	JButton	Изменить
private toolBar	JToolBar	Панель инструментов
private dataSpecialisation	JTable	Таблица
private textSearch	JTextField	Поле поискового запроса
private search	JButton	Поиск
private scroll	JScrollPane	Скролл
private specialisationService	SpecialisationService	Сервис Профессий
private t1	Thread	Поток 1 отвечает за редактирование данных
private t2	Thread	Поток 2 отвечает за формирование отчета
private static log	Logger	Логгер класса
private addDialogSpecialisation	AddDialogSpecialisation	Диалог добавления данных
private editDialogSpecialisation	EditDialogSpecialisation	Диалог редактирования данных

Описание методов класса представлено в таблице 41.

Таблица 41 - описание методов класса SpecialisationWindow

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод отображения окна
public editR	void	String[] arr — данные, полученные от пользователя	Вспомогательный метод изменения данных в таблице
public addR	void	String name— данные, полученные от пользователя	Вспомогательный метод добавления данных в таблицу

WorkerWindow.java

Класс приложения, визуализирующий экранную форму с рабочими

Описание полей класса представлено в таблице 42.

Таблица 42 - описание полей класса WorkerWindow

Название	Тип данных	Семантика
private window	JFrame	Окно приложения
private model	DefaultTableModel	Модель таблицы
private add	JButton	Добавить
private delete	JButton	Удалить
private edit	JButton	Изменить
private print	JButton	Печать
private work	JButton	Работа
private toolBar	JToolBar	Панель инструментов
private dataWorkers	JTable	Таблица
private textSearch	JTextField	Поле поискового запроса
private search	JButton	Поиск
private scroll	JScrollPane	Скролл
private employeeService	EmployeeService	Сервис Рабочих
private specialisationService	SpecialisationService	Сервис Профессий
private t1	Thread	Поток 1 отвечает за редактирование данных
private t2	Thread	Поток 2 отвечает за формирование отчета
private static log	Logger	Логгер класса
private addDialogSpecialisation	AddDialogSpecialisation	Диалог добавления данных
private editDialogSpecialisation	EditDialogSpecialisation	Диалог редактирования данных

Описание методов класса представлено в таблице 43.

Таблица 43 - описание методов классаWotkerWindow

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод отображения окна
public checkList	void		Метод проверки списка на отсутсвие записей
public makeXml	void		Метод загрузки данных в xml

public getSpecs	String[]		Вспоогательный метод получения строк всех профессий
public editR	void	String[] arr — данные, полученные от пользователя	Вспомогательный метод изменения данных в таблице
public addR	void	String[] arr— данные, полученные от пользователя	Вспомогательный метод добавления данных в таблицу

ManagerWindow.java

Класс приложения, визуализирующий экранную форму с менеджерами

Описание полей класса представлено в таблице 44.

Таблица 44 - описание полей класса ManagerWindow

Название	Тип данных	Семантика
private window	JFrame	Окно приложения
private model	DefaultTableModel	Модель таблицы
private add	JButton	Добавить
private delete	JButton	Удалить
private edit	JButton	Изменить
private print	JButton	Печать
private clients	JButton	Показать всех клиентов
private toolBar	JToolBar	Панель инструментов
private dataManagers	JTable	Таблица
private textSearch	JTextField	Поле поискового запроса
private search	JButton	Поиск
private scroll	JScrollPane	Скролл
private managerService	ManagerService	Сервис Менеджера
private t1	Thread	Поток 1 отвечает за редактирование данных
private t2	Thread	Поток 2 отвечает за формирование отчета
private static log	Logger	Логгер класса
private addDialogManager	AddDialogManager	Диалог добавления данных
private editDialogProd	EditDialogManager	Диалог редактирования данных

Описание методов класса представлено в таблице 45.

Таблица 45 - описание методов класса ManagerWindow

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод отображения окна
public checkList	void		Метод проверки списка на отсутсвие записей
public makeXml	void		Метод загрузки данных в xml
public editR	void	String[] arr — данные,	Вспомогательный

		полученные от пользователя	метод изменения данных в таблице
public addR	void	String[] arr— данные, полученные от пользователя	Вспомогательный метод добавления данных в таблицу

ClientWindow.java

Класс приложения, визуализирующий экранную форму с клиентами

Описание полей класса представлено в таблице 46.

Таблица 46 - описание полей класса ClientWindow

Название	Тип данных	Семантика
private window	JFrame	Окно приложения
private model	DefaultTableModel	Модель таблицы
private add	JButton	Добавить
private delete	JButton	Удалить
private edit	JButton	Изменить
private print	JButton	Печать
private contract	JButton	Кнопка контрактов
private toolBar	JToolBar	Панель инструментов
private dataClients	JTable	Таблица
private textSearch	JTextField	Поле поискового запроса
private search	JButton	Поиск
private scroll	JScrollPane	Скролл
private clientService	ClientService	Сервис Клиента
private t1	Thread	Поток 1 отвечает за редактирование данных
private t2	Thread	Поток 2 отвечает за формирование отчета
private static log	Logger	Логгер класса
private addDialogClient	AddDialogClient	Диалог добавления данных
private editDialogClient	EditDialogClient	Диалог редактирования данных

Описание методов класса представлено в таблице 47.

Таблица 47 - описание методов класса ClientWindow

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод отображения окна
public checkList	void		Метод проверки списка на отсутсвие записей
public makeXml	void		Метод загрузки данных в xml
public editR	void	String[] arr — данные,	Вспомогательный

		полученные от пользователя	метод изменения данных в таблице
public addR	void	String[] arr— данные, полученные от пользователя	Вспомогательный метод добавления данных в таблицу

ContractWindow.java

Класс приложения, визуализирующий экранную форму с договорами

Описание полей класса представлено в таблице 48.

Таблица 48 - описание полей класса ContractWindow

Название	Тип данных	Семантика
private clientID	int	Уникальный идентификтор клиента
private window	JFrame	Окно приложения
private model	DefaultTableModel	Модель таблицы
private add	JButton	Добавить
private delete	JButton	Удалить
private edit	JButton	Изменить
private print	JButton	Печать
private end	JButton	Завершить
private description	JButton	Показать описание
private toolBar	JToolBar	Панель инструментов
private dataContracts	JTable	Таблица
private textSearch	JTextField	Поле поискового запроса
private search	JButton	Поиск
private outdated	JButton	Показать просроченные договоры
private time	JButton	Показать договоры за указанный период
private month	JButton	Отчет за месяц
private drop	JButton	Сбросить
private scroll	JScrollPane	Скролл
private contractService	ContractService	Сервис Контракта
private managerService	ManagerService	Сервис Менеджера
private clientService	ClientService	Сервис Клиента
private t1	Thread	Поток 1 отвечает за редактирование данных
private t2	Thread	Поток 2 отвечает за формирование отчета
private static log	Logger	Логгер класса
private addDialogContract	AddDialogContract	Диалог добавления данных
private editDialogContract	EditDialogContract	Диалог редактирования данных

private dialogTimePeriodSelection	DialogTimePeriodSelection	Диалог изменения временного промежутка
private dialogMonthSelection	DialogMonthSelection	Диалог выбора месяца
private dateBegin	Date	Начало временного отрезка для сортировки
private dateEnd	Date	Конец временного отрезка для сортировки
private monthDate	Date	Месяц для отчета

Описание методов класса представлено в таблице 49.

Таблица 49 - описание методов класса ContractWindow

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод отображения окна
public checkList	void		Метод проверки списка на отсутсвие записей
public makeXml	void		Метод загрузки данных в xml
public editR	void	String[] arr — данные, полученные от пользователя	Вспомогательный метод изменения данных в таблице
public addR	void	String[] arr— данные, полученные от пользователя	Вспомогательный метод добавления данных в таблицу
public setDate	void	Date begin — начало временного отрезка Date end — конец временного отрезка	Вспомогательный метод установки даты для поиска
public setMonth	void	Date date — дата с месяцем	Вспомогательный метод установки месяца для отчета
public getManagers	String[]		Вспомогательный метод получения строк всех менеджеров
public getClients	String[]		Вспомогательный метод получения строк всех клиентов

WorkerContractWindow.java

Класс приложения, визуализирующий экранную форму с договорами, которые выполняет рабочий

Описание полей класса представлено в таблице 50.

Таблица 50 - описание полей класса WorkerContractWindow

Название	Тип данных	Семантика
private workerId	int	Уникальный идентификтор рабочего, для которого открыто окно
private window	JFrame	Окно приложения
private model	DefaultTableModel	Модель таблицы
private add	JButton	Добавить
private delete	JButton	Удалить
private edit	JButton	Изменить
private print	JButton	Печать
private description	JButton	Показать описание
private toolBar	JToolBar	Панель инструментов
private dataContracts	JTable	Таблица
private textSearch	JTextField	Поле поискового запроса
private search	JButton	Поиск
private outdated	JButton	Показать просроченные договоры
private time	JButton	Показать договоры за указанный период
private drop	JButton	Сбросить
private scroll	JScrollPane	Скролл
private workerService	EmployeeService	Сервис Рабочего
private contractrService	ContractService	Сервис Контракта
private t2	Thread	Поток 2 отвечает за формирование отчета
private static log	Logger	Логгер класса
private addDialog	AddDialogWorkerContract	Диалог добавления данных
private dialogTimePeriodSelection	DialogTimePeriodSelection	Диалог изменения временного промежутка
private dialogMonthSelection	DialogMonthSelection	Диалог выбора месяца
private dateBegin	Date	Начало временного отрезка
private dateEnd	Date	Конец временного отрезка

Описание методов класса представлено в таблице 51.

Таблица 51 - описание методов класса WorkerContraxtWindow

Название метода	Возвращаемый тип	Описание параметров	Назначение
public show	void		Метод отображения окна
public checkList	void		Метод проверки списка на отсутсвие записей
public makeXml	void		Метод загрузки данных в xml
public addR	void	int ID — данные, полученные от пользователя	Вспомогательный метод добавления данных в таблицу
public setDate	void	Date begin — начало временного отрезка Date end — конец временного отрезка	Вспомогательный метод установки даты для поиска

DialogSpecialisation.java

Абстрактный класс диалогового окна Добавления/Редактирования данных профессии

Описание полей класса представлено в таблице 52.

Таблица 52 - описание полей класса DialogSpecialisation

Название	Тип данных	Семантика
private name	JTextField	Текствове поле названия
private check	boolean	Переменная корректности ввода
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 53.

Таблица 53 - описание методов класса DialogSpecialisation

Название метода	Возвращаемый тип	Описание параметров	Назначение
public abstract progress	void	SpecialisationWindow parent — объект класса приложения	Выполнение манипуляций с данными
public abstract init	void	SpecialisationWindow parent — объект класса приложения	Инициализация
protected checher	void	JTextField field — проверяемое поле	Проверка поля на корректность введенных данных

AddDialogSpecialisation.java

Класс окна добавления данных профессии
Описание методов класса представлено в таблице 54.

Таблица 54 - описание методов класса AddDialogSpecialisation

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	SpecialisationWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	SpecialisationWindow parent — объект класса приложения	Инициализация

EditDialogSpecialisation.java

Класс окна изменения данных профессии
Описание методов класса представлено в таблице 55.

Таблица 55 - описание методов класса EditDialogSpecialisation

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	SpecialisationWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	SpecialisationWindow parent — объект класса приложения	Инициализация

DialogManager.java

Абстрактный класс диалогового окна Добавления/Редактирования данных менеджера

Описание полей класса представлено в таблице 56.

Таблица 56 - описание полей класса DialogManager

Название	Тип данных	Семантика
private name	JTextField	Текствове поле имени
private surname	JTextField	Текствове поле фамилии
private check	Boolean[]	Массив переменных, отвечающих за корректность ввода
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 57.

Таблица 57 - описание методов класса DialogManager

Название метода	Возвращаемый тип	Описание параметров	Назначение
public abstract progress	void	ManagerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public abstract init	void	ManagerWindow parent — объект класса приложения	Инициализация
protected checher	void	int i — номер поля JTextField field — проверяемое поле	Проверка поля на корректность введенных данных

AddDialogManager.java

Класс окна добавления данных менеджера
Описание методов класса представлено в таблице 58.

Таблица 58- описание методов класса AddDialogManager

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	ManagerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	ManagerWindow parent — объект класса приложения	Инициализация

EditDialogManager.java

Класс окна изменения данных менеджера
Описание методов класса представлено в таблице 59.

Таблица 59 - описание методов кlacd

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	ManagerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	ManagerWindow parent — объект класса приложения	Инициализация

DialogClient.java

Абстрактный класс диалогового окна Добавления/Редактирования данных клиента

Описание полей класса представлено в таблице 60.

Таблица 61 - описание полей класса DialogClient

Название	Тип данных	Семантика
private name	JTextField	Текствове поле имени
private surname	JTextField	Текствове поле фамилии
private company	JTextField	Текстовое поле компании
private check	Boolean[]	Массив переменных, отвечающих за корректность ввода
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 61.

Таблица 61 - описание методов класса DialgClient

Название метода	Возвращаемый тип	Описание параметров	Назначение
public abstract progress	void	ClientWindow parent — объект класса приложения	Выполнение манипуляций с данными
public abstract init	void	ClientWindow parent — объект класса приложения	Инициализация
protected checher	void	int i — номер поля JTextField field — проверяемое поле	Проверка поля на корректность введенных данных

AddDialogClient.java

Класс окна добавления данных клиента
Описание методов класса представлено в таблице 62.

Таблица 19 - описание методов класса AddDialogClient

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	ClientWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	ClientWindow parent — объект класса приложения	Инициализация

EditDialogClient.java

Класс окна изменения данных клиента
Описание методов класса представлено в таблице 63.

Таблица 63 - описание методов класса EditDialogClient

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	ClientWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	ClientWindow parent — объект класса приложения	Инициализация

DialogWorker.java

Абстрактный класс диалогового окна Добавления/Редактирования данных рабочего

Описание полей класса представлено в таблице 64.

Таблица 64 - описание полей класса DialogWindow

Название	Тип данных	Семантика
private name	JTextField	Текствове поле имени
private surname	JTextField	Текствове поле фамилии
private exp	JTextField	Текстовое поле опыта работы
private specs	JComboBox	Выподающий список профессии
private check	Boolean[]	Массив переменных, отвечающих за корректность ввода
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 65.

Таблица 65 - описание методов класса DialogWorker

Название метода	Возвращаемый тип	Описание параметров	Назначение
public abstract progress	void	WorkerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public abstract init	void	WorkerWindow parent — объект класса приложения	Инициализация
protected checher	void	int i — номер поля JTextField field — проверяемое поле	Проверка поля на корректность введенных данных

AddDialogWorker.java

Класс окна добавления данных рабочего
Описание методов класса представлено в таблице 66.

Таблица 66 - описание методов класса AddDialogWorker

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	WorkerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	WorkerWindow parent — объект класса приложения	Инициализация

EditDialogWorker.java

Класс окна изменения данных рабочего
Описание методов класса представлено в таблице 67.

Таблица 67 - описание методов класса EditDialogWorker

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	WorkerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	WorkerWindow parent — объект класса приложения	Инициализация

DialogContract.java

Абстрактный класс диалогового окна Добавления/Редактирования данных контракта

Описание полей класса представлено в таблице 68.

Таблица 68 - описание полей класса DialogContact

Название	Тип данных	Семантика
private description	JTextField	Текствове поле описания
private price	JTextField	Текствове поле цены
private clients	JComboBox	Выподающий список клиентов
private managers	JComboBox	Выподающий список менеджеров
private dataBegin	JDatePickerImpl	Дата пикер даты подписания контракта
private dataEnd	JDatePickerImpl	Дата пикер даты окончания контракта
private check	Boolean[]	Массив переменных, отвечающих за корректность ввода
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 69.

Таблица 69 - описание методов класса DialogContact

Название метода	Возвращаемый тип	Описание параметров	Назначение
public abstract progress	void	ContractWindow parent — объект класса приложения	Выполнение манипуляций с данными
public abstract init	void	ContractWindow parent — объект класса приложения	Инициализация
protected checher	void	int i — номер поля JTextField field — проверяемое поле	Проверка поля на корректность введенных данных

AddDialogContract.java

Класс окна добавления данных контракта
Описание методов класса представлено в таблице 70.

Таблица 19 - описание методов класса AddDialogContact

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	WorkerWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	WorkerWindow parent — объект класса приложения	Инициализация

EditDialogContract.java

Класс окна изменения данных контракта
Описание методов класса представлено в таблице 71.

Таблица 71 - описание методов класса EditDialogContact

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	ContractWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	ContractWindow parent — объект класса приложения	Инициализация

AddDialogWorkerContract.java

Класс диалогового окна добавления контрактов рабочему

Описание полей класса представлено в таблице 72.

Таблица 72 - описание полей класса AddDialogWorkerContract

Название	Тип данных	Семантика
private name	JTextField	Текствове поле названия
private check	boolean	Переменная корректности ввода
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 73.

Таблица 73 - описание методов класса AddDialogWorkerContract

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	WorkerContractWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void	WorkerContractWindow parent — объект класса приложения	Инициализация
protected checher	void	JTextField field — проверяемое поле	Проверка поля на корректность введенных данных

DialogMonthSelection.java

Класс диалогового окна получения временного периода для отображения данных договоров.

Описание полей класса представлено в таблице 74.

Таблица 74 - описание полей класса DialogMonthSelection

Название	Тип данны	Семантика
private dateBegin	JDatePickerImpl	Дата пикер начала временного отрезка
private dateEnd	JDatePickerImpl	Дата пикер конца временного отрезка
private ok	JButton	Кнопка принять
private cancel	JButton	Кнопка отменить

Описание методов класса представлено в таблице 75.

Таблица 75 - описание методов класса DialogMonthSelection

Название метода	Возвращаемый тип	Описание параметров	Назначение
public progress	void	ContractWindow parent — объект класса приложения	Выполнение манипуляций с данными
public init	void		Инициализация
public display	void		Показать интерфейс

Руководство оператору

Назначение программы

ПК «Администрирование завода по производству металлических изделий» должен входить в состав автоматизированной системы учёта и администрирования информации.

В рамках ПК «Администрирование завода по производству металлических изделий» администратор может:

- добавлять, править и удалять информацию о специализациях;
- добавлять, править и удалять информацию о рабочих;
- добавлять, править и удалять информацию о менеджерах;
- добавлять, править и удалять информацию о клиентах;
- добавлять, править и удалять информацию о договорах;
- получать отчёты о работе салона завода по производству металлических изделий;

Описание задачи

Разработать ПК (программный комплекс) для директора завода по производству металлических изделий. В ПК должна храниться информация о рабочих с указанием специализации, менеджерах, клиентах и договорах. Директор может добавлять, изменять и удалять информацию.

Обязательными требованиями при разработке кода ПК являются использование следующих конструкций языка Java:

- закрытые и открытые члены классов;
- наследование;
- конструкторы с параметрами;
- абстрактные базовые классы;
- виртуальные функции;
- обработка исключительных ситуаций;
- динамическое создание объектов.

На основании требований были разработаны программные классы.

Требования к коду ПК учтены при создании программных классов и непосредственном написании программы.

Входные и выходные данные

Выходные данные должны быть представлены в виде таблицы содержащей необходимые данные

Ввод исходных данных должен осуществляется в режиме диалога. Вводимые данные являются значениями характеристик информационных объектов. Вводимая информация может выбираться или набираться из списка предлагаемых значений.

Запуск программы

При запуске программы на экране

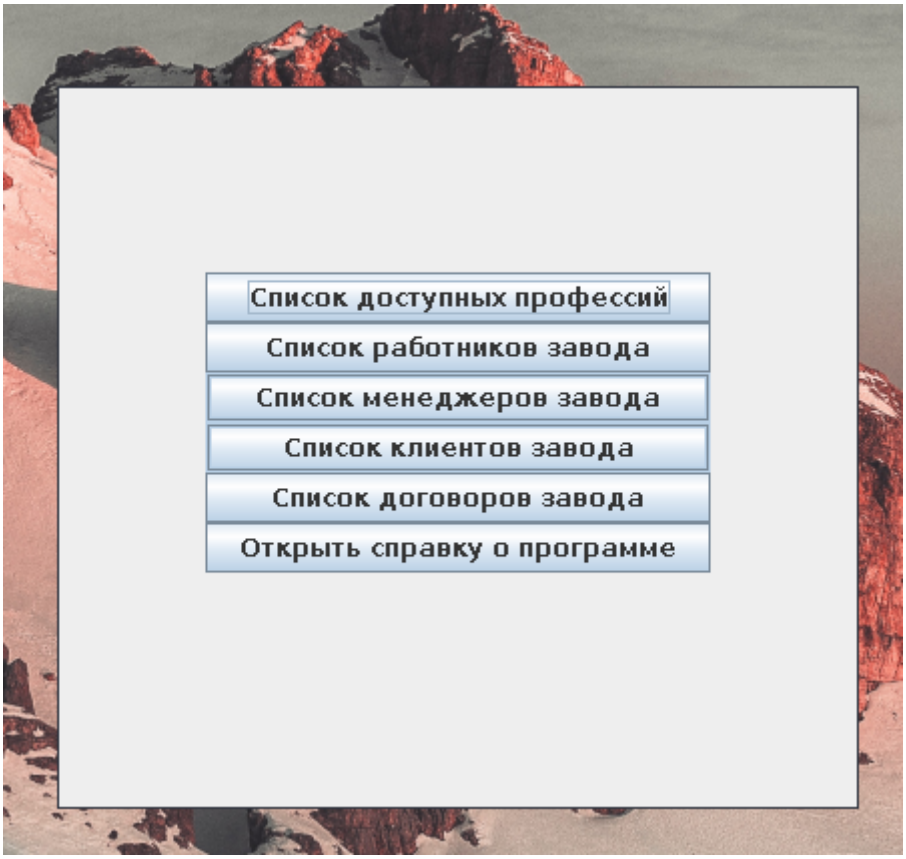


Рисунок 10: Главное меню

Чтоб показать нужну таблицу нажмите кнопку в главном меню. После этого откроется таблица,

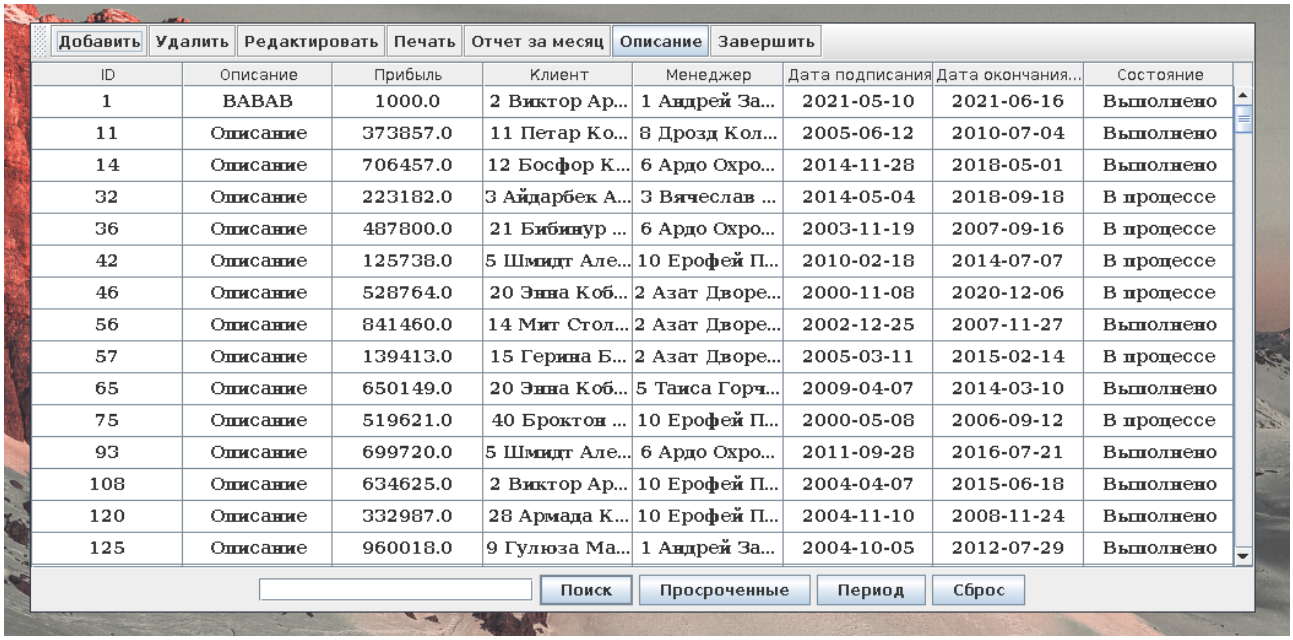


Рисунок 11: Таблица договоров

На главной панели будут доступны стандарные CRUD функции приложения, импорт таблицы в pdf и некоторые дополнительные функции для каждой таблицы. В нижней панели будет доступно поле поиска по разным полям БД и для таблицы с договорами некоторые кнопки сортировки.

Посмотрим, как происходит получение отчета о заводе. Чтоб получить отчет о работе завода за выбранный месяц нажмите кнопку на «Отчет за месяц». Перед вами всплывет следующее диалоговое окно.

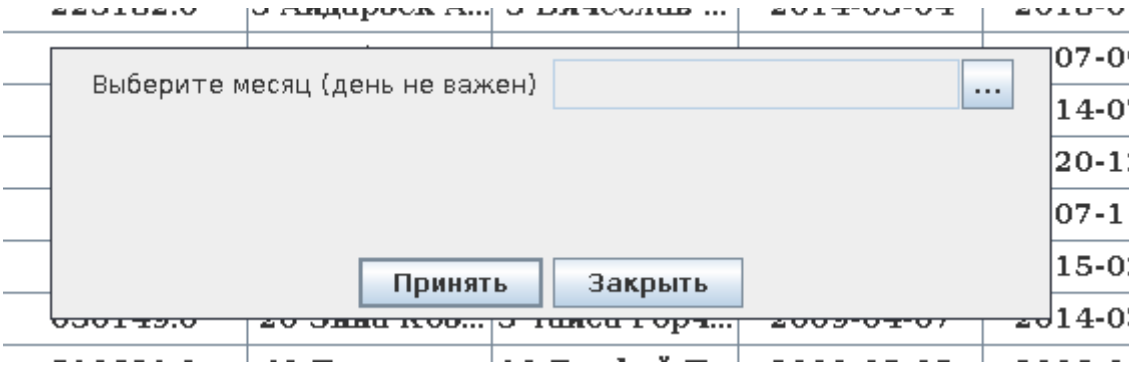


Рисунок 12: Диалог получения времени

Если нажмем на три точки, то сможем выбрать дату.

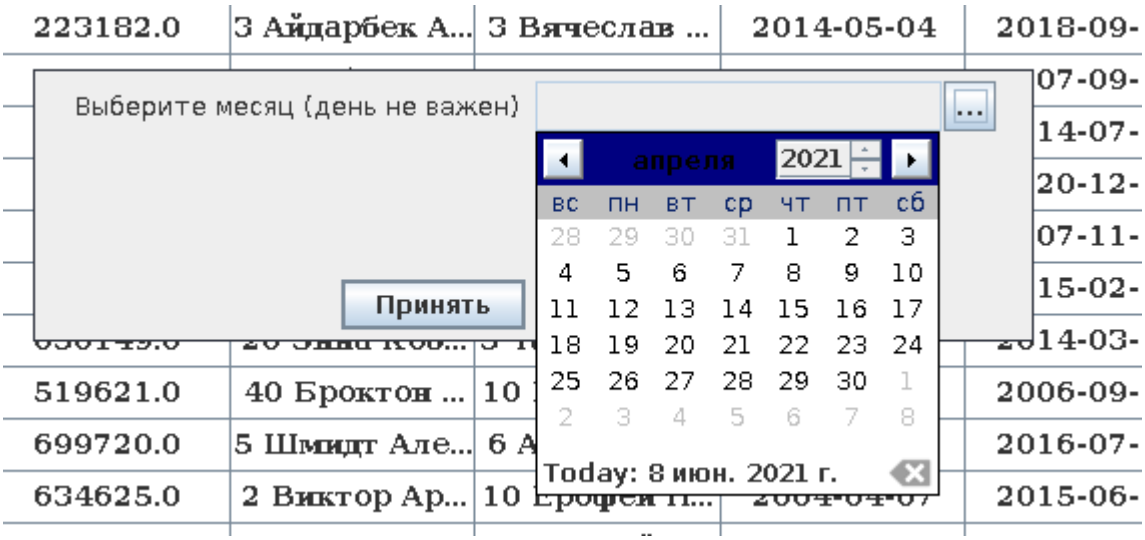


Рисунок 13: Поле выбора даты

После выбора даты и нажатия на кнопку принять получаем отчет о работе завода.

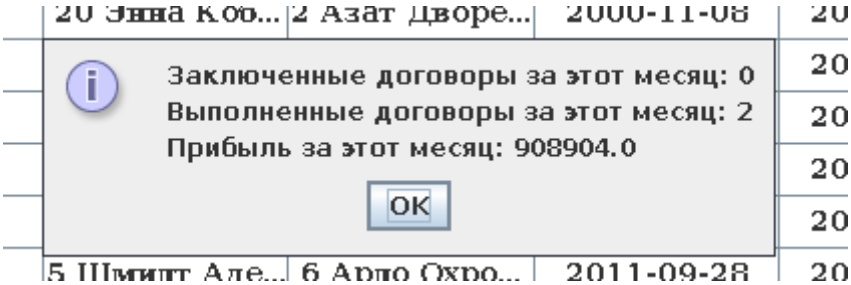


Рисунок 14: Отчет о работе завода

Описание базы данных

В курсовой работе была использована библиотека для Java hibernate для обращения к СУБД MySQL. Была создана база данных kurs и таблицы clients (для класса Client), employee (для класса Employee), managers (для класса Manager), specialisations (для класса Specialisation), contracts (для класса Contract) и таблица employees_contracts для организации связи ManyToMany между классами Employee и Contract. Поля баз таблиц и их связи между ними изображены на рисунке ниже.

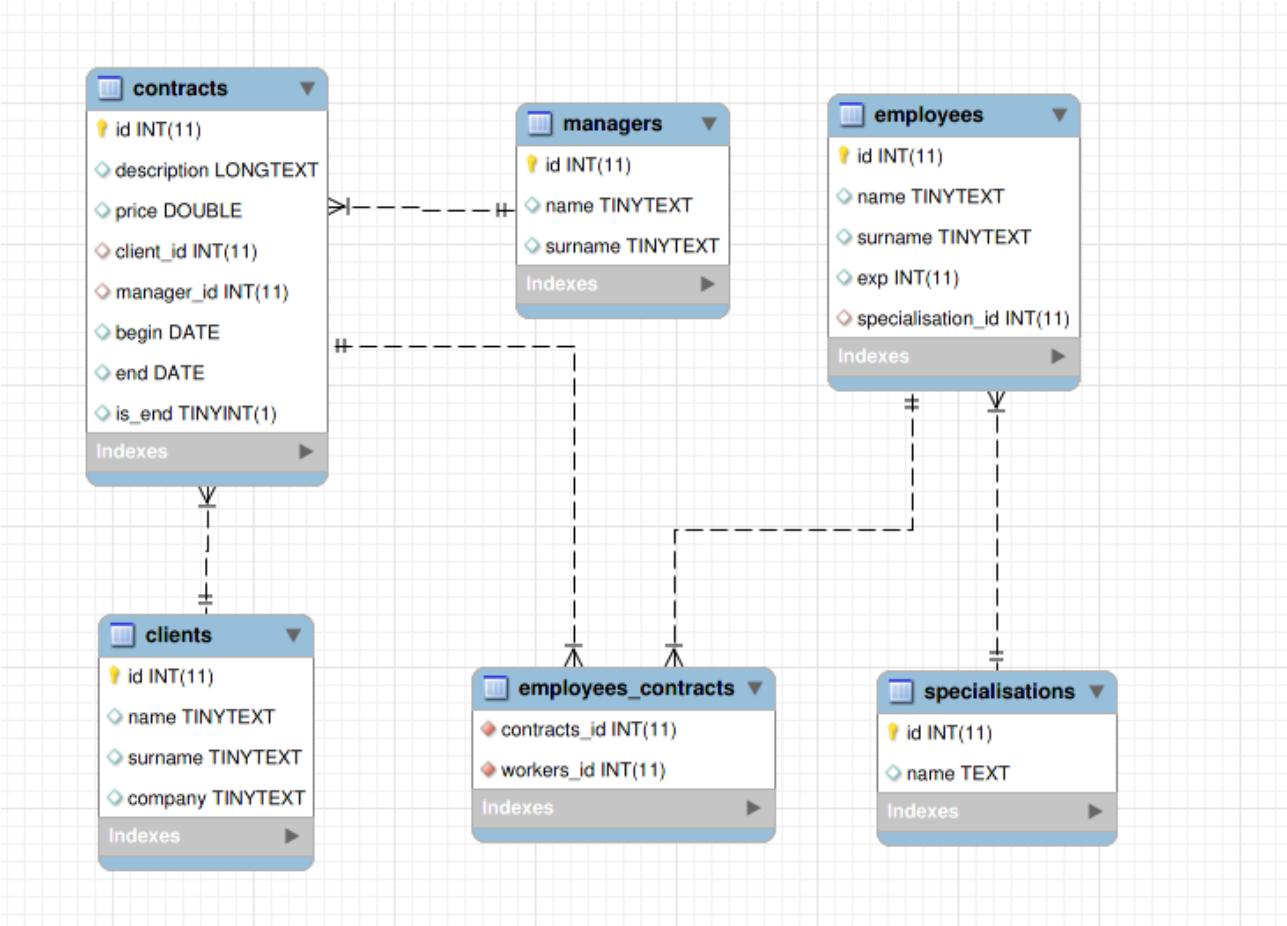


Рисунок 15: Схема базы данных

Заключение

В результате проделанной работы разработан ПК, предназначенный для администрирования работы завода по производству металлических изделий. В процессе проектирования ПК созданы прототип пользовательского графического интерфейса, диаграмма классов, руководство пользователю. Были закреплены теоретические знания, приобретены практические навыки по проектированию и разработке программного обеспечения на объектно-ориентированном языке Java.

ПРИЛОЖЕНИЕ А.

Исходный текст программы:

App.java

```
import Factory.gui.ContractWindow;
import Factory.gui.Menu;

import java.util.logging.Logger;

/** Главный класс, запускающий приложение */
public class App
{
    /** Логгер класса */
    private static final Logger log =
Logger.getLogger(ContractWindow.class.getName());

    /** Главный метод main */
    public static void main(String[] args)
    {
        log.info("Запуск программы");
        Menu m = new Menu();
        m.show();
    }
}
```

Package Factory.dao

ClientDAO.java

```
package Factory.dao;
import Factory.model.*;

import Factory.util.HibernateSessionFactoryUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

import java.util.List;

/** Data Access Object для класса Client */
public class ClientDAO
{
    /** Текущая сессия */
    private Session currentSession;
    /** Текущая транзакция */
    private Transaction currentTransaction;

    /** Стандартный конструктор */
```

```
    public ClientDAO() { }

    /** Открыть текущую сессию */
    public Session openCurrentSession()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        return currentSession;
    }

    /** Открыть текущую сессию с транзакцией */
    public Session openCurrentSessionwithTransaction()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        currentTransaction = currentSession.beginTransaction();
        return currentSession;
    }

    /** Закрыть текущую сессию */
    public void closeCurrentSession()
    {
        currentSession.close();
    }

    /** Закрыть текущую сессию с транзакцией */
    public void closeCurrentSessionwithTransaction()
    {
        currentTransaction.commit();
        currentSession.close();
    }

    /** Получить текущую сессию */
    public Session getCurrentSession() {
        return currentSession;
    }

    /** Установить текущую сессию */
    public void setCurrentSession(Session currentSession) {
        this.currentSession = currentSession;
    }

    /** Получить текущую транзакцию */
    public Transaction getCurrentTransaction() {
        return currentTransaction;
    }

    /** Установить текущую сессию */
    public void setCurrentTransaction(Transaction
currentTransaction) {
```

```
        this.currentTransaction = currentTransaction;
    }

    /** Добавить клиента в БД
     *
     * @param entity клиент
     */
    public void persist(Client entity) {
        getCurrentSession().save(entity);
    }

    /** Обновить клиента в БД
     *
     * @param entity клиент
     */
    public void update(Client entity) {
        getCurrentSession().update(entity);
    }

    /** Найти клиента в БД по уникальному идентификатору
     *
     * @param id уникальный идентификатор
     */
    public Client findById(int id)
    {
        return getCurrentSession().get(Client.class, id);
    }

    /** Удалить клиента из БД
     *
     * @param entity клиент
     */
    public void delete(Client entity) {
        getCurrentSession().delete(entity);
    }

    /** Найти всех клиентов в БД */
    @SuppressWarnings("unchecked")
    public List<Client> findAll() {
        return (List<Client>)
getCurrentSession().createQuery("SELECT c FROM clients c").list();
    }

    /** Найти всех клиентов указанного менеджера в БД
     * @param id - уникальный идентификатор менеджера
     */
    @SuppressWarnings("unchecked")
    public List<Client> findByManagerId(int id)
    {
        Query<Client> q = getCurrentSession().createQuery("SELECT
c FROM clients c JOIN c.client_contracts s WHERE s.manager.id
```

```
= :id");
        q.setParameter("id", id);
        return q.list();
    }

    /**Удалить всех клиентов из БД */
    public void deleteAll() {
        List<Client> entityList = findAll();
        for (Client entity : entityList)
        {
            delete(entity);
        }
    }

}
```

ContractDAO.java

```
package Factory.dao;
import Factory.model.*;

import Factory.util.HibernateSessionFactoryUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

import java.util.Date;
import java.util.List;

/** Data Access Object для класса Client */
public class ContractDAO
{
    /** Текущая сессия */
    private Session currentSession;
    /** Текущая транзакция */
    private Transaction currentTransaction;

    /** Стандартный конструктор */
    public ContractDAO() { }

    /** Открыть текущую сессию */
    public Session openCurrentSession()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        return currentSession;
    }

    /** Открыть текущую сессию с транзакцией */
    public Session openCurrentSessionwithTransaction()
    {
        currentSession =
```

```
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        currentTransaction = currentSession.beginTransaction();
        return currentSession;
    }

    /** Закрыть текущую сессию */
    public void closeCurrentSession()
    {
        currentSession.close();
    }

    /** Закрыть текущую сессию с транзакцией */
    public void closeCurrentSessionwithTransaction()
    {
        currentTransaction.commit();
        currentSession.close();
    }

    /** Получить текущую сессию */
    public Session getCurrentSession() {
        return currentSession;
    }

    /** Установить текущую сессию */
    public void setCurrentSession(Session currentSession) {
        this.currentSession = currentSession;
    }

    /** Получить текущую транзакцию */
    public Transaction getCurrentTransaction() {
        return currentTransaction;
    }

    /** Установить текущую сессию */
    public void setCurrentTransaction(Transaction
currentTransaction) {
        this.currentTransaction = currentTransaction;
    }

    /** Добавить контракт в БД
     *
     * @param entity клиент
     */
    public void persist(Contract entity) {
        getCurrentSession().save(entity);
    }

    /** Обновить контракт в БД
     *
     * @param entity клиент
```

```
     */
    public void update(Contract entity) {
        getCurrentSession().update(entity);
    }

    /** Найти контракт в БД по уникальному идентификатору
     *
     * @param id уникальный идентификатор
     * @return найденные контракты
     */
    public Contract findById(int id)
    {
        return getCurrentSession().get(Contract.class, id);
    }

    /** Удалить контракт из БД
     *
     * @param entity клиент
     */
    public void delete(Contract entity) {
        getCurrentSession().delete(entity);
    }

    /** Найти всех контракты в БД
     * @return найденные контракты
     */
    @SuppressWarnings("unchecked")
    public List<Contract> findAll() {
        return (List<Contract>)
getCurrentSession().createQuery("SELECT c FROM contracts
c").list();
    }

    /** Найти все устаревшие контракты в БД
     * @return найденные контракты
     */
    @SuppressWarnings("unchecked")
    public List<Contract> findOutdated()
    {
        return (List<Contract>)
getCurrentSession().createQuery("FROM contracts c WHERE c.isEnd =
0 and c.dateEnd <= current_date").list();
    }

    /** Найти все новые контракты в БД
     * @param date дата, по которой ищем в бд
     * @return найденные контракты
     */
    public int findNew(Date date)
    {
        Query<Contract> q =
```

```
getCurrentSession().createQuery("SELECT c FROM contracts c WHERE
month(c.dateBegin) = month(:date) and year(c.dateBegin) =
year(:date)");
        q.setParameter("date", date);
        return q.list().size();
    }

    /** Найти все завершенные контракты в БД
     * @param date дата, по которой ищем в бд
     * @return найденные контракты
     */
    public List<Contract> findFinish(Date date)
    {
        Query<Contract> q =
getCurrentSession().createQuery("SELECT c FROM contracts c WHERE
month(c.dateEnd) = month(:date) and year(c.dateEnd) = year(:date)
and c.isEnd = 1");
        q.setParameter("date", date);
        return q.list();
    }

    /** Найти все устаревшие контракты в БД
     * @param id уникальный идентификатор рабочего
     * @return найденные контракты
     */
    public List<Contract> findOutdated(int id)
    {
        Query<Contract> q =
getCurrentSession().createQuery("SELECT c FROM contracts c JOIN
c.workers w WHERE w.id = :id and c.isEnd = 0 and c.dateEnd <=
current_date");
        q.setParameter("id", id);
        return q.list();
    }

    /** Найти все контракты указанного контракт в БД
     * @param id уникальный идентификатор контракт
     * @return найденные контракты
     */
    public List<Contract> findByClientId(int id)
    {
        Query<Contract> q =
getCurrentSession().createQuery("SELECT c FROM contracts c WHERE
c.client.id = :id");
        q.setParameter("id", id);
        return q.list();
    }

    /** Найти период времени
     *
     * @param btime начало промежутка
```

```
     * @param etime конец промежутка
     * @return найденные контракты
     */
    @SuppressWarnings("unchecked")
    public List<Contract> findTimePeriod(Date btime, Date etime)
    {
        Query<Contract> q = getCurrentSession().createQuery("FROM
contracts c WHERE c.dateBegin BETWEEN :btime AND :etime");
        q.setParameter("btime", btime);
        q.setParameter("etime", etime);
        return q.list();
    }

    /** Найти период времени
     *
     * @param btime начало промежутка
     * @param etime конец промежутка
     * @param id уникальный идентификатор
     * @return найденные контракты
     */
    @SuppressWarnings("unchecked")
    public List<Contract> findTimePeriod(Date btime, Date etime,
int id)
    {
        Query<Contract> q =
getCurrentSession().createQuery("SELECT c FROM contracts c JOIN
c.workers w WHERE w.id = :id AND c.dateBegin BETWEEN :btime
AND :etime");
        q.setParameter("id", id);
        q.setParameter("btime", btime);
        q.setParameter("etime", etime);
        return q.list();
    }

    /**Удалить всех контракты из БД */
    public void deleteAll() {
        List<Contract> entityList = findAll();
        for (Contract entity : entityList)
        {
            delete(entity);
        }
    }

}
```

EmployeeDAO.java

```
package Factory.dao;
import Factory.model.*;

import Factory.util.HibernateSessionFactoryUtil;
import org.hibernate.Session;
```

```
import org.hibernate.Transaction;
import java.util.List;

public class EmployeeDAO
{
    private Session currentSession;
    private Transaction currentTransaction;

    public EmployeeDAO() { }

    public Session openCurrentSession()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        return currentSession;
    }

    public Session openCurrentSessionwithTransaction()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        currentTransaction = currentSession.beginTransaction();
        return currentSession;
    }

    public void closeCurrentSession()
    {
        currentSession.close();
    }

    public void closeCurrentSessionwithTransaction()
    {
        currentTransaction.commit();
        currentSession.close();
    }

    public Session getCurrentSession() {
        return currentSession;
    }

    public void setCurrentSession(Session currentSession) {
        this.currentSession = currentSession;
    }

    public Transaction getCurrentTransaction() {
        return currentTransaction;
    }

    public void setCurrentTransaction(Transaction
currentTransaction) {
```

```
        this.currentTransaction = currentTransaction;
    }

    public void persist(Employee entity) {
        getCurrentSession().save(entity);
    }

    public void update(Employee entity) {
        getCurrentSession().update(entity);
    }

    public Employee findById(int id)
    {
        return getCurrentSession().get(Employee.class, id);
    }

    public void delete(Employee entity) {
        getCurrentSession().delete(entity);
    }

    @SuppressWarnings("unchecked")
    public List<Employee> findAll() {
        return (List<Employee>)
getCurrentSession().createQuery("SELECT e FROM employees
e").list();
    }

    public void deleteAll() {
        List<Employee> entityList = findAll();
        for (Employee entity : entityList)
        {
            delete(entity);
        }
    }

}
```

ManagerDAO.java

```
package Factory.dao;
import Factory.model.*;

import Factory.util.HibernateSessionFactoryUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;
import java.util.List;

public class ManagerDAO
{
    private Session currentSession;
    private Transaction currentTransaction;
```

```
    public ManagerDAO() { }

    public Session openCurrentSession()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        return currentSession;
    }

    public Session openCurrentSessionwithTransaction()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        currentTransaction = currentSession.beginTransaction();
        return currentSession;
    }

    public void closeCurrentSession()
    {
        currentSession.close();
    }

    public void closeCurrentSessionwithTransaction()
    {
        currentTransaction.commit();
        currentSession.close();
    }

    public Session getCurrentSession() {
        return currentSession;
    }

    public void setCurrentSession(Session currentSession) {
        this.currentSession = currentSession;
    }

    public Transaction getCurrentTransaction() {
        return currentTransaction;
    }

    public void setCurrentTransaction(Transaction
currentTransaction) {
        this.currentTransaction = currentTransaction;
    }

    public void persist(Manager entity) {
        getCurrentSession().save(entity);
    }

    public void update(Manager entity) {
        getCurrentSession().update(entity);
    }
```

```
    public Manager findById(int id)
    {
        return getCurrentSession().get(Manager.class, id);
    }

    public void delete(Manager entity) {
        getCurrentSession().delete(entity);
    }

    @SuppressWarnings("unchecked")
    public List<Manager> findAll() {
        return (List<Manager>)
getCurrentSession().createQuery("SELECT m FROM managers
m").list();
    }

    public void deleteAll() {
        List<Manager> entityList = findAll();
        for (Manager entity : entityList) {
            delete(entity);
        }
    }

}
```

SpecialisationDAO.java

```
package Factory.dao;
import Factory.model.*;

import javax.persistence.Query;

import Factory.util.HibernateSessionFactoryUtil;
import org.hibernate.Session;
import org.hibernate.Transaction;
import java.util.List;

public class SpecialisationDAO
{
    private Session currentSession;
    private Transaction currentTransaction;

    public SpecialisationDAO() { }

    public Session openCurrentSession()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        return currentSession;
    }
```

```
    public Session openCurrentSessionwithTransaction()
    {
        currentSession =
HibernateSessionFactoryUtil.getSessionFactory().openSession();
        currentTransaction = currentSession.beginTransaction();
        return currentSession;
    }

    public void closeCurrentSession()
    {
        currentSession.close();
    }

    public void closeCurrentSessionwithTransaction()
    {
        currentTransaction.commit();
        currentSession.close();
    }

    public Session getCurrentSession() {
        return currentSession;
    }

    public void setCurrentSession(Session currentSession) {
        this.currentSession = currentSession;
    }

    public Transaction getCurrentTransaction() {
        return currentTransaction;
    }

    public void setCurrentTransaction(Transaction
currentTransaction) {
        this.currentTransaction = currentTransaction;
    }

    public void persist(Specialisation entity) {
        getCurrentSession().save(entity);
    }

    public void update(Specialisation entity) {
        getCurrentSession().update(entity);
    }

    public Specialisation findById(int id)
    {
        return getCurrentSession().get(Specialisation.class, id);
    }

    public Specialisation findByName(String name)
    {
        Query query = getCurrentSession().createQuery("FROM
```

```
specialisations WHERE name = :name");
        query.setParameter("name", name);
        return (Specialisation) query.getSingleResult();
    }

    public void delete(Specialisation entity) {
        getCurrentSession().delete(entity);
    }

    @SuppressWarnings("unchecked")
    public List<Specialisation> findAll() {
        return (List<Specialisation>)
getCurrentSession().createQuery("SELECT s FROM specialisations
s").list();
    }

    public void deleteAll() {
        List<Specialisation> entityList = findAll();
        for (Specialisation entity : entityList) {
            delete(entity);
        }
    }

}
```

Package Factory.exceptions

UnacceptableTimePeriod.java

```
package Factory.exceptions;

/**
 * Класс исключения.
 * Запрет на запись в файл пустого списка
 */
public class UnacceptableTimePeriod extends Exception {

    /** Конструктор ошибки */
    public UnacceptableTimePeriod ()
    {
        super("Некорректный временной промежуток. Попробуйте
заного");
    }
}
```

EmptyFileException.java

```
package Factory.exceptions;

/**
 * Класс исключения.
 * Запрет на запись в файл пустого списка
 * @author Яловега Никита 9308
 * @version 0.6
 */
public class EmptyFileException extends Exception {

    /** Конструктор ошибки */
    public EmptyFileException ()
    {
        super("Запись в файл невозможна! Список пуст");
    }
}
```

Package Factory.gui

Menu.java

```
package Factory.gui;

import javax.swing.*;
import java.awt.*;

/** Класс главного меню */
public class Menu
{
    /** Метод показа главного меню */
    public void show()
    {
        JFrame window = new JFrame("factory: Главное окно");
        JButton specialisationWindow = new JButton("Список
доступных профессий");
        JButton workerWindow = new JButton("Список работников
завода"); // , new ImageIcon("./img/prodList.png")
        JButton managerWindow = new JButton("Список менеджеров
завода");
        JButton clientWindow = new JButton("Список клиентов
завода");
        JButton contractWindow = new JButton("Список договоров
завода");
        JButton refWindow = new JButton("Открыть справку о
программе");

        specialisationWindow.addActionListener((e) -> {
            try {
                new SpecialisationWindow();
```

```
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка: " +
ex);
            }
        });

        workerWindow.addActionListener((e) -> {
            try {
                new WorkerWindow();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка: " +
ex);
            }
        });

        managerWindow.addActionListener((e) -> {
            try {
                new ManagerWindow();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка: " +
ex);
            }
        });
        clientWindow.addActionListener((e) -> {
            try {
                new ClientWindow();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка: " +
ex);
            }
        });
        contractWindow.addActionListener((e) -> {
            try {
                new ContractWindow();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Ошибка: " +
ex);
            }
        });

        refWindow.addActionListener((e) -> new Ref(window,
"Справка"));

        window.setLayout(new GridBagLayout());
        JPanel mainp = new JPanel();

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(7, 1));

        panel.setSize(350, 100);

        panel.add(specialisationWindow);
```

```
        panel.add(workerWindow);
        panel.add(managerWindow);
        panel.add(clientWindow);
        panel.add(contractWindow);
        panel.add(refWindow);
        mainp.add(panel);
        window.add(mainp);
        window.setSize(400, 380);
        window.setLocation(500, 200);
        window.setVisible(true);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Ref.java

```
package Factory.gui;

import javax.swing.*;
import java.awt.*;

/**
 * Класс окна справки
 */
public class Ref extends JDialog
{
    /** Контруктор окна справки
     *
     * @param owner - окно, от которого запускалось текущее окно
     * @param title - название окна
     */
    public Ref(JFrame owner, String title)
    {
        super(owner,title,true);
        setLayout(new FlowLayout(FlowLayout.CENTER,5,5));
        JLabel text = new JLabel();
        text.setText("<html>ПК директора завода по производству
металлических изделий <br><br>Разработчик: Яловега Никита
Владиславович 9308</html>");
        ImageIcon icon = new ImageIcon("img/cat.png");
        Image image = icon.getImage();
        Image newimg = image.getScaledInstance(200, 200,
java.awt.Image.SCALE_SMOOTH);
        icon = new ImageIcon(newimg);
        JLabel label = new JLabel();
        label.setIcon(icon);
        add(text,BorderLayout.WEST);
        add(label, BorderLayout.EAST);
        setSize(520,300);
        setLocation(400,250);
        setVisible(true);
    }
```

```
}
```

SpecialisationWindow.java

```
package Factory.gui;

import Factory.model.*;
import Factory.service.*;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.util.List;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с профессиями
*/
public class SpecialisationWindow
{
    /** Стандартный конструктор */
    SpecialisationWindow()
    {
        show();
    }

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;

    /** Изменить */
    private JButton edit;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataSpecialisations;

    /** Поле поискового запроса */
    private JTextField textSearch;
```

```
    /** Поиск */
    private JButton search;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис Менеджера */
    private SpecialisationService specialisationService = new
SpecialisationService();

    /** Поток 1 отвечает за редактирование данных */
    Thread t1 = new Thread();

    /** Поток 2 отвечает за формирование отчет */
    Thread t2 = new Thread();

    /** Логгер класса */
    private static final Logger log =
Logger.getLogger(SpecialisationWindow.class.getName());

    /** Диалог добавления данных */
    private AddDialogSpecialisation addDialogSpecialisation;

    /** Диалог измения данных */
    private EditDialogSpecialisation editDialogSpecialisation;

    /** Метод отображения окна */
    public void show()
    {
        log.info("Открытие окна SpecialisationWindow");
        window = new JFrame("factory: Список менеджеров завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        // Создание кнопок и прикрепление иконок
        log.info("Добавление кнопок к окну SpecialisationWindow");
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        edit = new JButton("Редактировать");

        // Настройка подсказок
        add.setToolTipText("Добавить информацию о рабочих");
        delete.setToolTipText("Удалить информацию о рабочих");
        edit.setToolTipText("Изменить информацию о рабочих");
        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(edit);
        // Размещение панели инструментов
```

```
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);
        // Создание таблицы с данными
        log.info("Добавление таблицы с данными к окну
SpecialisationWindow");
        String[] columns = {"ID", "Название должности"};

        List<Specialisation> specialisationsList =
specialisationService.findAll();
        String [][] data = new String[specialisationsList.size()]
[5];
        for (int i = 0; i < specialisationsList.size(); i++)
        {
            data[i] = specialisationsList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataSpecialisations = new JTable(model);
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
        dataSpecialisations.setRowSorter(sort);
        dataSpecialisations.setFont(new
Font(Font.SERIF,Font.BOLD,14));
        dataSpecialisations.setIntercellSpacing(new
Dimension(0,1));

dataSpecialisations.setRowHeight(dataSpecialisations.getRowHeight(
)+10);

dataSpecialisations.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUM
NS);


dataSpecialisations.setDefaultRenderer(dataSpecialisations.getColu
mnClass(1), new DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
```

```
            }

        });

        scroll = new JScrollPane(this.dataSpecialisations);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        textSearch = new JTextField();
        textSearch.setColumns(20);
        search = new JButton("Поиск");
        window.getRootPane().setDefaultButton(search);
// remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);

        // Если не выделена строка, то прячем кнопки

dataSpecialisations.getSelectionModel().addListSelectionListener((
e) -> {
            boolean check = !
dataSpecialisations.getSelectionModel().isSelectionEmpty();
            edit.setVisible(check);
            delete.setVisible(check);
        });

        add.addActionListener((e) -> {
            log.info("Старт Add listener");
            addDialogSpecialisation = new
AddDialogSpecialisation(window, SpecialisationWindow.this,
"Добавление записи");
            addDialogSpecialisation.setVisible(true);
        });
```

```
        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
            if (dataSpecialisations.getRowCount() > 0) {
                if (dataSpecialisations.getSelectedRow() != -1) {
                    try {

specialisationService.delete(Integer.parseInt(dataSpecialisations.
getValueAt(dataSpecialisations.getSelectedRow(), 0).toString()));

model.removeRow(dataSpecialisations.convertRowIndexToModel(dataSpe
cialisations.getSelectedRow()));
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    } catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });

        delete.setMnemonic(KeyEvent.VK_D);

        edit.addActionListener((e)-> {
            log.info("Старт Edit listener");
            if (model.getRowCount() != 0) {
                if (dataSpecialisations.getSelectedRow() != -1) {
                    t1 = new Thread(() -> {
                        JOptionPane.showMessageDialog(null,"1
поток запущен");
                        editDialogSpecialisation = new
EditDialogSpecialisation(window, SpecialisationWindow.this,
"Редактирование");
                        editDialogSpecialisation.setVisible(true);
                    });
                    t1.start();
                } else {
                    JOptionPane.showMessageDialog(null, "Не
```

```
выбрана строка. Нечего редактировать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для редактирования");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего редактировать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего редактировать");
            }
        });
        edit.setMnemonic(KeyEvent.VK_E);

        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataSpecialisations.setRowSorter(sorter);
            }
        });

        window.setVisible(true);
    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param name - данные, полученные от пользователя
     */
    public void addR(String name)
    {
        Specialisation newS = new Specialisation(name);
        specialisationService.persist(newS);
        model.addRow(newS.toTableFormat());
    }
```

```
    /**
     * Вспомогательный метод изменения данных в таблице
     * @param arr - данные, полученные от пользователя
     */
    public void editR(String[] arr)
    {
        Specialisation M =
specialisationService.findById(Integer.parseInt(arr[0]));
        M.setName(arr[1]);
        specialisationService.update(M);
    }

}
```

WorkerWindow.java

```
package Factory.gui;

import Factory.exceptions.EmptyFileException;
import Factory.model.*;
import Factory.service.*;

import Factory.util.ReportUtil;
import org.w3c.dom.*;

import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.List;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с рабочими */
public class WorkerWindow
{
    /** Стандартный конструктор */
    WorkerWindow()
    {
```

```
        show();
    }

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;

    /** Изменить */
    private JButton edit;

    /** Печать */
    private JButton print;

    /** Работа */
    private JButton work;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataWorkers;

    /** Поле поискового запроса */
    private JTextField textSearch;

    /** Поиск */
    private JButton search;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис Рабочего */
    private EmployeeService employeeService = new
EmployeeService();

    /** Сервис Профессий */
    private SpecialisationService specialisationService = new
SpecialisationService();

    /** Поток 1 отвечает за редактирование данных */
    Thread t1 = new Thread();
```

```
    /** Поток 2 отвечает за формирование отчет */
    Thread t2 = new Thread();

    /** Логгер класса */
    private static final Logger log =
Logger.getLogger(ManagerWindow.class.getName());

    /** Диалог добавления данных */
    private AddDialogWorker addDialogWorker;

    /** Диалог измения данных */
    private EditDialogWorker editDialogWorker;

    /** Метод отображения окна */
    public void show(){
        log.info("Открытие окна WorkerWindow");
        window = new JFrame("factory: Список рабочих завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        // Создание кнопок и прикрепление иконок
        log.info("Добавление кнопок к окну WorkerWindow");
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        edit = new JButton("Редактировать");
        print = new JButton("Печать");
        work = new JButton("Работа");

        // Настройка подсказок
        add.setToolTipText("Добавить информацию о рабочих");
        delete.setToolTipText("Удалить информацию о рабочих");
        edit.setToolTipText("Изменить информацию о рабочих");
        print.setToolTipText("Распечатать информацию о рабочих");
        work.setToolTipText("Показать договоры, которые выполняют
рабочие");
        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(edit);
        toolBar.add(print);
        toolBar.add(work);
        // Размещение панели инструментов
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);
        // Создание таблицы с данными
        log.info("Добавление таблицы с данными к окну
WorkerWindow");
        String[] columns = {"ID", "Имя", "Фамилия", "Опыт работы",
"Должность"};
```

```
        List<Employee> workersList = employeeService.findAll();
        String [][] data = new String[workersList.size()][5];
        for (int i = 0; i < workersList.size(); i++)
        {
            data[i] = workersList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataWorkers = new JTable(model);
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
        dataWorkers.setRowSorter(sort);
        dataWorkers.setFont(new Font(Font.SERIF,Font.BOLD,14));
        dataWorkers.setIntercellSpacing(new Dimension(0,1));
        dataWorkers.setRowHeight(dataWorkers.getRowHeight()+10);

dataWorkers.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);


dataWorkers.setDefaultRenderer(dataWorkers.getColumnClass(1), new
DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
            }


        });

        scroll = new JScrollPane(this.dataWorkers);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        // Подготовка компонентов поиска
        textSearch = new JTextField();
        textSearch.setColumns(20);
```

```
        search = new JButton("Поиск");
        window.getRootPane().setDefaultButton(search);
        // remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
        // retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);

        // Если не выделена строка, то прячем кнопки

dataWorkers.getSelectionModel().addListSelectionListener((e) -> {
            boolean check = !
dataWorkers.getSelectionModel().isSelectionEmpty();
            edit.setVisible(check);
            delete.setVisible(check);
            work.setVisible(check);
        });

        add.addActionListener((e) ->
        {
            log.info("Старт Add listener");
            addDialogWorker = new AddDialogWorker(window,
WorkerWindow.this, "Добавление записи");
            addDialogWorker.setVisible(true);
        });

        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
            if (dataWorkers.getRowCount() > 0) {
                if (dataWorkers.getSelectedRow() != -1) {
                    try {

employeeService.delete(Integer.parseInt(dataWorkers.getValueAt(dat
aWorkers.getSelectedRow(), 0).toString()));

model.removeRow(dataWorkers.convertRowIndexToModel(dataWorkers.get
SelectedRow()));
```

```
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    } catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });

        delete.setMnemonic(KeyEvent.VK_D);

        edit.addActionListener((e)-> {
            log.info("Старт Edit listener");
            if (model.getRowCount() != 0) {
                if (dataWorkers.getSelectedRow() != -1) {
                    t1 = new Thread(() -> {
                        JOptionPane.showMessageDialog(null,"1
поток запущен");
                        editDialogWorker = new
EditDialogWorker(window, WorkerWindow.this, "Редактирование");
                        editDialogWorker.setVisible(true);
                    });
                    t1.start();
                } else {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего редактировать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего редактировать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });
        edit.setMnemonic(KeyEvent.VK_E);
```

```
        print.addActionListener((e)->{
            log.info("Старт Print listener");
            t2 = new Thread(() -> {
                try {
                    JFileChooser fileChooser = new JFileChooser();
                    fileChooser.setDialogTitle("Select where you
want to save the report");
                    FileFilter pdf = new
FileNameExtensionFilter("PDF file(.pdf)", "pdf");
                    fileChooser.addChoosableFileFilter(pdf);
                    fileChooser.setCurrentDirectory(new
File("."));
                    String resultpath = "reportWorkers.pdf";
                    int returnVal =
fileChooser.showSaveDialog(null);
                    if(returnVal == JFileChooser.APPROVE_OPTION)
                    {
                        File file = fileChooser.getSelectedFile();
                        resultpath = file.getAbsolutePath();
                    }
                    checkList();
                    makeXml();
                    ReportUtil.print("dataWorkers.xml",
"window/dataWorkers", "workers.jrxml", resultpath);
                    JOptionPane.showMessageDialog(null, "2 поток
закончил работу. Отчет создан");
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Ошибка: "
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            });
            t2.start();
        });

        work.addActionListener((e) -> {
            log.info("Старт work listener");
            if (dataWorkers.getRowCount() > 0) {
                if (dataWorkers.getSelectedRow() != -1) {
                    try
                    {
                        new
WorkerContractWindow(Integer.parseInt(dataWorkers.getValueAt(dataW
orkers.getSelectedRow(), 0).toString()));
                    }
                    catch (Exception ex)
                    {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
```

```
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка");
                }
            }
            else
            {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });


        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataWorkers.setRowSorter(sorter);
            }
        });

        window.setVisible(true);
    }

    /**
     * Метод проверки списка на отсутсвие записей
     * @throws EmptyFileException моё исключение
     */
    private void checkList() throws EmptyFileException
    {
```

```
        if(model.getRowCount() == 0)
            throw new EmptyFileException();
    }

    /** Метод загрузки данных в XML файл */
    public void makeXml()
    {
        try
        {
            // Создание парсера документа
            DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
            // Создание пустого документа
            Document doc = builder.newDocument();
            // Создание корневого элемента window и добавление его
в документ
            Node window = doc.createElement("window");
            doc.appendChild(window);
            // Создание дочерних элементов dataEmploy и присвоение
значений атрибутам
            for (int i = 0; i < model.getRowCount(); i++) {
                Element dataEmploy =
doc.createElement("dataWorkers");
                window.appendChild(dataEmploy);
                dataEmploy.setAttribute("name", (String)
model.getValueAt(i, 1));
                dataEmploy.setAttribute("surname", (String)
model.getValueAt(i, 2));
                dataEmploy.setAttribute("exp", (String)
model.getValueAt(i, 3));
                dataEmploy.setAttribute("specialisation", (String)
model.getValueAt(i, 4));
            }
            try
            {
                // Создание преобразователя документа
                Transformer trans =
TransformerFactory.newInstance().newTransformer();
                // Создание файла с именем dataEmploy.xml для
записи документа
                java.io.FileWriter fw = new
FileWriter("dataWorkers.xml");
                // Запись документа в файл
                trans.transform(new DOMSource(doc), new
StreamResult(fw));
            } catch (TransformerConfigurationException e) {
                e.printStackTrace();
            } catch (TransformerException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
```

```
            }

        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }


    /** Вспомогательный метод получения строк всех профессий */
    public String[] getSpecs()
    {
        List<Specialisation> specs =
specialisationService.findAll();
        String [] result = new String[specs.size()];
        for (int i = 0; i < specs.size(); i++)
            result[i] = specs.get(i).getName();
        return result;
    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param arr - данные, полученные от пользователя
     */
    public void addR(String[] arr)
    {
        Employee newW = new Employee(arr[0], arr[1],
Integer.parseInt(arr[2]),
specialisationService.findByName(arr[3]));
        employeeService.persist(newW);
        model.addRow(newW.toTableFormat());
    }

    /**
     * Вспомогательный метод изменения данных в таблице
     * @param arr - данные, полученные от пользователя
     */
    public void editR(String[] arr)
    {
        Employee W =
employeeService.findById(Integer.parseInt(arr[0]));
        W.setName(arr[1]);
        W.setSurname(arr[2]);
        W.setExp(Integer.parseInt(arr[3]));

W.setSpecialisation(specialisationService.findByName(arr[4]));
        employeeService.update(W);
    }
}
```

ManagerWindow.java

```java
package Factory.gui;

import Factory.exceptions.EmptyFileException;
import Factory.model.*;
import Factory.service.*;
import Factory.util.ReportUtil;

import org.w3c.dom.*;

import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.List;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с менеджерами
*/
public class ManagerWindow
{
    /** Стандартный конструктор */
    ManagerWindow()
    {
        show();
    }

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;
```

```
    /** Изменить */
    private JButton edit;

    /** Печать */
    private JButton print;

    /** Показать всех клиентов */
    private JButton clients;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataManagers;

    /** Поле поискового запроса */
    private JTextField textSearch;

    /** Поиск */
    private JButton search;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис Менеджера */
    private ManagerService managerService = new ManagerService();

    /** Поток 1 отвечает за редактирование данных */
    Thread t1 = new Thread();
    /** Поток 3 отвечает за формирование отчет */
    Thread t2 = new Thread();

    /** Логгер класса */
    private static final Logger log =
Logger.getLogger(ManagerWindow.class.getName());

    /** Диалог добавления данных */
    private AddDialogManager addDialogManager;

    /** Диалог измения данных */
    private EditDialogManager editDialogProd;

    /** Метод отображения окна */
    public void show()
    {
        log.info("Открытие окна ManagerWindow");
        window = new JFrame("factory: Список менеджеров завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
```

```
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        // Создание кнопок и прикрепление иконок
        log.info("Добавление кнопок к окну ManagerWindow");
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        edit = new JButton("Редактировать");
        print = new JButton("Печать");
        clients = new JButton("Клиенты");

        // Настройка подсказок
        add.setToolTipText("Добавить информацию о менеджерах");
        delete.setToolTipText("Удалить информацию о менеджерах");
        edit.setToolTipText("Изменить информацию о менеджерах");
        print.setToolTipText("Распечатать информацию о
менеджерах");
        clients.setToolTipText("Показать клиентов менеджера");

        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(edit);
        toolBar.add(print);
        toolBar.add(clients);

        // Размещение панели инструментов
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);
        // Создание таблицы с данными
        String[] columns = {"ID", "Имя", "Фамилия"};
        log.info("Добавление таблицы с данными к окну
ManagerWindow");
        List<Manager> managersList = managerService.findAll();
        String [][] data = new String[managersList.size()][5];
        for (int i = 0; i < managersList.size(); i++)
        {
            data[i] = managersList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataManagers = new JTable(model);
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
```

```
        dataManagers.setRowSorter(sort);
        dataManagers.setFont(new Font(Font.SERIF,Font.BOLD,14));
        dataManagers.setIntercellSpacing(new Dimension(0,1));
        dataManagers.setRowHeight(dataManagers.getRowHeight()+10);

dataManagers.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);


dataManagers.setDefaultRenderer(dataManagers.getColumnClass(1),
new DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
            }

        });

        scroll = new JScrollPane(this.dataManagers);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        // Подготовка компонентов поиска
        textSearch = new JTextField();
        textSearch.setColumns(20);
        search = new JButton("Поиск");
        window.getRootPane().setDefaultButton(search);
        // remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
        // retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);
```

```
        // Если не выделена строка, то прячем кнопки

dataManagers.getSelectionModel().addListSelectionListener((e) -> {
            boolean check = !
dataManagers.getSelectionModel().isSelectionEmpty();
            edit.setVisible(check);
            delete.setVisible(check);
        });

        add.addActionListener((e) -> {
            log.info("Старт Add listener");
            addDialogManager = new AddDialogManager(window,
ManagerWindow.this, "Добавление записи");
            addDialogManager.setVisible(true);
        });

        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
            if (dataManagers.getRowCount() > 0) {
                if (dataManagers.getSelectedRow() != -1) {
                    try {

managerService.delete(Integer.parseInt(dataManagers.getValueAt(dat
aManagers.getSelectedRow(), 0).toString()));

model.removeRow(dataManagers.convertRowIndexToModel(dataManagers.g
etSelectedRow()));
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    } catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,
"Ошибка" + ex.toString());
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });
```

```
        delete.setMnemonic(KeyEvent.VK_D);

        edit.addActionListener((e)-> {
            log.info("Старт Edit listener");
            if (model.getRowCount() != 0) {
                if (dataManagers.getSelectedRow() != -1) {
                    t1 = new Thread(() -> {
                        JOptionPane.showMessageDialog(null,"1
поток запущен");
                        editDialogProd = new
EditDialogManager(window, ManagerWindow.this, "Редактирование");
                        editDialogProd.setVisible(true);
                    });
                    t1.start();
                } else {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего редактировать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего редактировать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });
        edit.setMnemonic(KeyEvent.VK_E);

        print.addActionListener((e)->{
            log.info("Старт Print listener");
            t2 = new Thread(() -> {
                try {
                    JFileChooser fileChooser = new JFileChooser();
                    fileChooser.setDialogTitle("Select where you
want to save the report");
                    FileFilter pdf = new
FileNameExtensionFilter("PDF file(.pdf)", "pdf");
                    fileChooser.addChoosableFileFilter(pdf);
                    fileChooser.setCurrentDirectory(new
File("."));
                    String resultpath = "reportManagers.pdf";
                    int returnVal =
fileChooser.showSaveDialog(null);
                    if(returnVal == JFileChooser.APPROVE_OPTION)
                    {
                        File file = fileChooser.getSelectedFile();
                        resultpath = file.getAbsolutePath();
                    }
                    checkList();
                    makeXml();
```

```
                    ReportUtil.print("dataManagers.xml",
"window/dataManagers", "managers.jrxml", resultpath);
                    JOptionPane.showMessageDialog(null, "2 поток
закончил работу. Отчет создан");
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Ошибка: "
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            });
            t2.start();
        });

        clients.addActionListener(e -> {
            log.info("Старт clients listener");
            if (model.getRowCount() != 0) {
                if (dataManagers.getSelectedRow() != -1) {
                    try
                    {
                        new
ClientWindow(Integer.parseInt(dataManagers.getValueAt(dataManagers
.getSelectedRow(), 0).toString()));
                    }
                    catch (Exception ex)
                    {
                        JOptionPane.showMessageDialog(null,
"Ошибка" + ex.toString());
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего показывать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка");
                }
            } else
            {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего показывать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего показывать");
            }
        });

        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
```

```
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataManagers.setRowSorter(sorter);
            }
        });

        window.setVisible(true);
    }

    /**
     * Метод проверки списка на отсутсвие записей
     * @throws EmptyFileException моё исключение
     */
    private void checkList() throws EmptyFileException
    {
        if(model.getRowCount() == 0)
            throw new EmptyFileException();
    }

    /** Метод загрузки данных в XML файл */
    public void makeXml() {
        try {
            // Создание парсера документа
            DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
            // Создание пустого документа
            Document doc = builder.newDocument();
            // Создание корневого элемента window и добавление его
в документ
            Node window = doc.createElement("window");
            doc.appendChild(window);
            // Создание дочерних элементов dataEmploy и присвоение
значений атрибутам
            for (int i = 0; i < model.getRowCount(); i++)
            {
                Element dataManager =
doc.createElement("dataManagers");
                window.appendChild(dataManager);
                dataManager.setAttribute("name", (String)
model.getValueAt(i, 1));
```

```
                dataManager.setAttribute("surname", (String)
model.getValueAt(i, 2));
            }
            try {
                // Создание преобразователя документа
                Transformer trans =
TransformerFactory.newInstance().newTransformer();
                // Создание файла с именем dataEmploy.xml для
записи документа
                java.io.FileWriter fw = new
FileWriter("dataManagers.xml");
                // Запись документа в файл
                trans.transform(new DOMSource(doc), new
StreamResult(fw));
            } catch (TransformerConfigurationException e) {
                e.printStackTrace();
            } catch (TransformerException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }


        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param arr - данные, полученные от пользователя
     */
    public void addR(String[] arr)
    {
        Manager newM = new Manager(arr[0], arr[1]);
        managerService.persist(newM);
        model.addRow(newM.toTableFormat());
    }

    /**
     * Вспомогательный метод изменения данных в таблице
     * @param arr - данные, полученные от пользователя
     */
    public void editR(String[] arr)
    {
        Manager M =
managerService.findById(Integer.parseInt(arr[0]));
        M.setName(arr[1]);
        M.setSurname(arr[2]);
        managerService.update(M);
    }
```

```
}
```

ClientWindow.java

```
package Factory.gui;

import Factory.exceptions.EmptyFileException;
import Factory.model.*;
import Factory.service.*;

import Factory.util.ReportUtil;
import org.w3c.dom.*;

import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import javax.swing.*;
import javax.swing.table.*;
import javax.swing.filechooser.FileFilter;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.List;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с клиентами
*/
public class ClientWindow
{
    /** Стандартный конструктор */
    ClientWindow()
    {
        this.managerID = -1;
        show();
    }

    /**
     * Конструктор для показа определенных клиентов
     * @param id - уникальный идентификатор менеджера
     */
    ClientWindow(int id)
    {
```

```
        this.managerID = id;
        show();
    }

    /** Идентификатор менеджера, для которого вызвано окно */
    private int managerID;

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;

    /** Изменить */
    private JButton edit;

    /** Печать */
    private JButton print;

    /** Кнопка контрактов */
    private JButton contract;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataClients;

    /** Поле поискового запроса */
    private JTextField textSearch;

    /** Поиск */
    private JButton search;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис Менеджера */
    private ClientService clientService = new ClientService();

    /** Поток 1 отвечает за редактирование данных */
    Thread t1 = new Thread();
```

```
    /** Поток 3 отвечает за формирование отчет */
    Thread t2 = new Thread();

    /** Логгер класса */
    private static final Logger log =
Logger.getLogger(ClientWindow.class.getName());

    /** Диалог добавления данных */
    private AddDialogClient addDialogClient;

    /** Диалог измения данных */
    private EditDialogClient editDialogClient;

    /** Метод отображения окна */
    public void show()
    {
        log.info("Открытие окна ManagerWindow");
        window = new JFrame("factory: Список клиентов завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        log.info("Добавление кнопок к окну ClientWindow");
        // Создание кнопок и прикрепление иконок
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        edit = new JButton("Редактировать");
        print = new JButton("Печать");
        contract = new JButton("Контракты");

        // Настройка подсказок
        add.setToolTipText("Добавить информацию о клиентах");
        delete.setToolTipText("Удалить информацию о клиентах");
        edit.setToolTipText("Изменить информацию о клиентах");
        print.setToolTipText("Распечатать информацию о клиентах");
        contract.setToolTipText("Показать контракты клиентов");
        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(edit);
        toolBar.add(print);
        toolBar.add(contract);
        // Размещение панели инструментов
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);
        // Создание таблицы с данными
        String[] columns = {"ID", "Имя", "Фамилия", "Компания"};
        log.info("Добавление таблицы с данными к окну
ClientWindow");
        List<Client> clientsList;
        if (managerID == -1)
```

```
            clientsList = clientService.findAll();
        else
            clientsList =
clientService.findByManagerId(managerID);

        String [][] data = new String[clientsList.size()][4];
        for (int i = 0; i < clientsList.size(); i++)
        {
            data[i] = clientsList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataClients = new JTable(model);
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
        dataClients.setRowSorter(sort);
        dataClients.setFont(new Font(Font.SERIF,Font.BOLD,14));
        dataClients.setIntercellSpacing(new Dimension(0,1));
        dataClients.setRowHeight(dataClients.getRowHeight()+10);

dataClients.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);


dataClients.setDefaultRenderer(dataClients.getColumnClass(1), new
DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
            }


        });

        // Если не выделена строка, то прячем кнопки

dataClients.getSelectionModel().addListSelectionListener((e) -> {
            boolean check = !
dataClients.getSelectionModel().isSelectionEmpty();
            edit.setVisible(check);
```

```
            delete.setVisible(check);
            contract.setVisible(check);
        });

        scroll = new JScrollPane(this.dataClients);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        // Подготовка компонентов поиска
        textSearch = new JTextField();
        textSearch.setColumns(20);
        search = new JButton("Поиск");
        window.getRootPane().setDefaultButton(search);
        // remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
        // retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);

        add.addActionListener((e) -> {
            log.info("Старт Add listener");
            addDialogClient = new AddDialogClient(window,
ClientWindow.this, "Добавление записи");
            addDialogClient.setVisible(true);
            log.info("Завершение Add listener");
        });

        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
            if (dataClients.getRowCount() > 0) {
                if (dataClients.getSelectedRow() != -1) {
                    try {

clientService.delete(Integer.parseInt(dataClients.getValueAt(dataC
lients.getSelectedRow(), 0).toString()));
```

```
model.removeRow(dataClients.convertRowIndexToModel(dataClients.get
SelectedRow()));
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    } catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });

        delete.setMnemonic(KeyEvent.VK_D);

        edit.addActionListener((e)-> {
            log.info("Старт Edit listener");
            if (model.getRowCount() != 0) {
                if (dataClients.getSelectedRow() != -1) {
                    t1 = new Thread(() -> {
                        JOptionPane.showMessageDialog(null,"1
поток запущен");
                        editDialogClient = new
EditDialogClient(window, ClientWindow.this, "Редактирование");
                        editDialogClient.setVisible(true);
                    });
                    t1.start();
                } else {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего редактировать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего редактировать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });
```

```
        edit.setMnemonic(KeyEvent.VK_E);

        print.addActionListener((e)->{
            log.info("Старт Print listener");
            t2 = new Thread(() -> {
                try {
                    JFileChooser fileChooser = new JFileChooser();
                    fileChooser.setDialogTitle("Select where you
want to save the report");
                    FileFilter pdf = new
FileNameExtensionFilter("PDF file(.pdf)", "pdf");
                    fileChooser.addChoosableFileFilter(pdf);
                    fileChooser.setCurrentDirectory(new
File("."));
                    String resultpath = "reportClients.pdf";
                    int returnVal =
fileChooser.showSaveDialog(null);
                    if(returnVal == JFileChooser.APPROVE_OPTION)
                    {
                        File file = fileChooser.getSelectedFile();
                        resultpath = file.getAbsolutePath();
                    }
                    checkList();
                    makeXml();
                    ReportUtil.print("dataClients.xml",
"window/dataClients", "clients.jrxml", resultpath);
                    JOptionPane.showMessageDialog(null, "2 поток
закончил работу. Отчет создан");
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Ошибка: "
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            });
            t2.start();
        });

        print.setMnemonic(KeyEvent.VK_E);

        contract.addActionListener((e)-> {
            log.info("Старт contract listener");
            if (model.getRowCount() != 0) {
                if (dataClients.getSelectedRow() != -1)
                {
                    try
                    {
                        new
ContractWindow(Integer.parseInt(dataClients.getValueAt(dataClients
.getSelectedRow(), 0).toString()));
                    }
                    catch (Exception ex) {
```

```
                        JOptionPane.showMessageDialog(null,
"Ошибка:" + ex.toString());
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                }
                else
                {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего показывать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего показывать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });
        contract.setMnemonic(KeyEvent.VK_E);

        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataClients.setRowSorter(sorter);
            }
        });

        window.setVisible(true);
    }

    /**
     * Метод проверки списка на отсутсвие записей
     * @throws EmptyFileException моё исключение
```

```
     */
    private void checkList() throws EmptyFileException
    {
        if(model.getRowCount() == 0)
            throw new EmptyFileException();
    }

    /** Метод загрузки данных в XML файл */
    public void makeXml() {
        try {
            // Создание парсера документа
            DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
            // Создание пустого документа
            Document doc = builder.newDocument();
            // Создание корневого элемента window и добавление его
в документ
            Node window = doc.createElement("window");
            doc.appendChild(window);
            // Создание дочерних элементов dataEmploy и присвоение
значений атрибутам
            for (int i = 0; i < model.getRowCount(); i++)
            {
                Element dataManager =
doc.createElement("dataClients");
                window.appendChild(dataManager);
                dataManager.setAttribute("name", (String)
model.getValueAt(i, 1));
                dataManager.setAttribute("surname", (String)
model.getValueAt(i, 2));
                dataManager.setAttribute("company", (String)
model.getValueAt(i, 3));
            }
            try {
                // Создание преобразователя документа
                Transformer trans =
TransformerFactory.newInstance().newTransformer();
                // Создание файла с именем dataEmploy.xml для
записи документа
                java.io.FileWriter fw = new
FileWriter("dataClients.xml");
                // Запись документа в файл
                trans.transform(new DOMSource(doc), new
StreamResult(fw));
            } catch (TransformerConfigurationException e) {
                e.printStackTrace();
            } catch (TransformerException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param arr - данные, полученные от пользователя
     */
    public void addR(String[] arr)
    {
        Client newC = new Client(arr[0], arr[1], arr[2]);
        clientService.persist(newC);
        model.addRow(newC.toTableFormat());
    }

    /**
     * Вспомогательный метод изменения данных в таблице
     * @param arr - данные, полученные от пользователя
     */
    public void editR(String[] arr)
    {
        Client C =
clientService.findById(Integer.parseInt(arr[0]));
        C.setName(arr[1]);
        C.setSurname(arr[2]);
        C.setCompany(arr[3]);
        clientService.update(C);
    }
}
```

ContractWindow.java

```
package Factory.gui;

import Factory.model.*;
import Factory.service.*;
import Factory.exceptions.*;

import Factory.util.ReportUtil;
import org.w3c.dom.*;

import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
```

```java
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.List;
import java.util.Date;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с договорами
*/
public class ContractWindow
{
    /** Стандартный конструктор */
    ContractWindow()
    {
        this.clientID = -1;
        show();
    }

    /**
     * Конструктор с указанием клиента
     * @param id - уникальный идентификатор клиента
     */
    ContractWindow(int id)
    {
        this.clientID = id;
        show();
    }

    /** Уникальный идентификатор клиента */
    private int clientID;

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;

    /** Изменить */
    private JButton edit;
```

```
    /** Печать */
    private JButton print;

    /** Завершить */
    private JButton end;

    /** Показать описание */
    private JButton description;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataContracts;

    /** Поле поискового запроса */
    private JTextField textSearch;

    /** Поиск */
    private JButton search;

    /** Показать просроченные договоры */
    private JButton outdated;

    /** Показать договоры за указанный период */
    private JButton time;

    /** Отчет за месяц */
    private JButton month;

    /** Сбросить */
    private JButton drop;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис Контрактов */
    private ContractService contractService = new
ContractService();

    /** Сервис Менеджеров */
    private ManagerService managerService = new ManagerService();

    /** Сервис Клиентов */
    private ClientService clientService = new ClientService();

    /** Поток 1 отвечает за редактирование данных */
    Thread t1 = new Thread();
```

```
    /** Поток 2 отвечает за формирование отчет */
    Thread t2 = new Thread();

    /** Логгер класса */
    private static final Logger log =
Logger.getLogger(ContractWindow.class.getName());

    /** Диалог добавления данных */
    private AddDialogContract addDialogContract;

    /** Диалог измения данных */
    private EditDialogContract editDialogContract;

    /** Диалог изменения временного промежутка */
    private DialogTimePeriodSelection dialogTimePeriodSelection;

    /** Диалог выбора месяца */
    private DialogMonthSelection dialogMonthSelection;

    /** Начало временного отрезка для сортировки */
    private Date dateBegin;

    /** Конец временного отрезка для сортировки */
    private Date dateEnd;

    /** Месяц для отчета */
    private Date monthDate;

    public void show()
    {
        log.info("Открытие окна ContractWindow");

        window = new JFrame("factory: Список договоров завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        // Создание кнопок и прикрепление иконок
        log.info("Добавление кнопок к окну ContractWindow");
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        edit = new JButton("Редактировать");
        print = new JButton("Печать");
        month = new JButton("Отчет за месяц");
        description = new JButton("Описание");
        end = new JButton("Завершить");



        // Настройка подсказок
        add.setToolTipText("Добавить контракт");
```

```
        delete.setToolTipText("Удалить контракт");
        edit.setToolTipText("Изменить контракт");
        print.setToolTipText("Распечатать контракты");
        month.setToolTipText("Отчет за месяц");
        description.setToolTipText("Показать описание контракта");
        end.setToolTipText("Завершить выполнение контракта");


        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(edit);
        toolBar.add(print);
        toolBar.add(month);
        toolBar.add(description);
        toolBar.add(end);
        // Размещение панели инструментов
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);

        // Создание таблицы с данными
        String[] columns = {"ID", "Описание", "Прибыль", "Клиент",
"Менеджер", "Дата подписания", "Дата окончания работ",
"Состояние"};
        log.info("Добавление таблицы с данными к окну
ContractWindow");
        List<Contract> contractsList;

        if (clientID == -1)
            contractsList = contractService.findAll();
        else
            contractsList =
contractService.findByClientId(clientID);

        String [][] data = new String[contractsList.size()][5];
        for (int i = 0; i < contractsList.size(); i++)
        {
            data[i] = contractsList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataContracts = new JTable(model);
```

```
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
        dataContracts.setRowSorter(sort);
        dataContracts.setFont(new Font(Font.SERIF,Font.BOLD,14));
        dataContracts.setIntercellSpacing(new Dimension(0,1));
        dataContracts.setRowHeight(dataContracts.getRowHeight()
+10);

dataContracts.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);


dataContracts.setDefaultRenderer(dataContracts.getColumnClass(1),
new DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
            }

        });

        // Если не выделена строка, то прячем кнопки

dataContracts.getSelectionModel().addListSelectionListener((e) ->
{
            boolean check = !
dataContracts.getSelectionModel().isSelectionEmpty();
            edit.setVisible(check);
            delete.setVisible(check);
            end.setVisible(check);
            description.setVisible(check);
        });

        scroll = new JScrollPane(this.dataContracts);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        // Подготовка компонентов поиска
        textSearch = new JTextField();
        textSearch.setColumns(20);
        search = new JButton("Поиск");
        outdated = new JButton("Просроченные");
        time = new JButton("Период");
        drop = new JButton("Сброс");
        outdated.setToolTipText("Показать просроченные договоры");
        time.setToolTipText("Показать договоры за определенный
```

```
промежуток времени");
        window.getRootPane().setDefaultButton(search);
        // remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
        // retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);
        searchPanel.add(outdated);
        searchPanel.add(time);
        searchPanel.add(drop);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);

        add.addActionListener((e) -> {
            log.info("Старт Add listener");
            addDialogContract = new AddDialogContract(window,
ContractWindow.this, "Добавление записи");
            addDialogContract.setVisible(true);
        });

        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
            if (dataContracts.getRowCount() > 0) {
                if (dataContracts.getSelectedRow() != -1) {
                    try {

contractService.delete(Integer.parseInt(dataContracts.getValueAt(d
ataContracts.getSelectedRow(), 0).toString()));

model.removeRow(dataContracts.convertRowIndexToModel(dataContracts
.getSelectedRow()));
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    } catch (Exception ex) {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
```

```
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });

        delete.setMnemonic(KeyEvent.VK_D);

        edit.addActionListener((e)-> {
            log.info("Старт Edit listener");
            if (model.getRowCount() != 0) {
                if (dataContracts.getSelectedRow() != -1) {
                    t1 = new Thread(() -> {
                        editDialogContract = new
EditDialogContract(window, ContractWindow.this, "Редактирование");
                        editDialogContract.setVisible(true);
                    });
                    t1.start();
                } else {
                    JOptionPane.showMessageDialog(null, "Не
выбрана строка. Нечего редактировать");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для редактирования");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего редактировать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего редактировать");
            }
        });
        edit.setMnemonic(KeyEvent.VK_E);

        end.addActionListener((e) -> {
            log.info("Старт End listener");
            if (dataContracts.getRowCount() > 0) {
                if (dataContracts.getSelectedRow() != -1) {
                    try
                    {

contractService.setEnd(Integer.parseInt(dataContracts.getValueAt(d
ataContracts.getSelectedRow(), 0).toString()));
                        dataContracts.setValueAt("Выполнено"
```

```
,dataContracts.getSelectedRow(), 7);
                        JOptionPane.showMessageDialog(window, "Вы
отметили договор завершеным");
                    } catch (Exception ex)
                    {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали договор для завершения");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для завершения");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего завершать");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего завершать");
            }
        });

        end.setMnemonic(KeyEvent.VK_D);

        description.addActionListener((e) -> {
            log.info("Старт Description listener");
            if (dataContracts.getRowCount() > 0) {
                if (dataContracts.getSelectedRow() != -1) {
                    try
                    {
                        JOptionPane.showMessageDialog(window,
dataContracts.getValueAt(dataContracts.getSelectedRow(),
1).toString());
                    }
                    catch (Exception ex)
                    {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали договор");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
```

```
            }
        });

        description.setMnemonic(KeyEvent.VK_D);

        print.addActionListener((e)->{
            log.info("Старт Print listener");
            t2 = new Thread(() -> {
                try {
                    JFileChooser fileChooser = new JFileChooser();
                    fileChooser.setDialogTitle("Select where you
want to save the report");
                    FileFilter pdf = new
FileNameExtensionFilter("PDF file(.pdf)", "pdf");
                    fileChooser.addChoosableFileFilter(pdf);
                    fileChooser.setCurrentDirectory(new
File("."));
                    String resultpath = "reportContracts.pdf";
                    int returnVal =
fileChooser.showSaveDialog(null);
                    if(returnVal == JFileChooser.APPROVE_OPTION)
                    {
                        File file = fileChooser.getSelectedFile();
                        resultpath = file.getAbsolutePath();
                    }
                    checkList();
                    makeXml();
                    ReportUtil.print("dataContracts.xml",
"window/dataContracts", "contracts.jrxml", resultpath);
                    JOptionPane.showMessageDialog(null, "2 поток
закончил работу. Отчет создан");
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Ошибка: "
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            });
            t2.start();
        });

        print.setMnemonic(KeyEvent.VK_D);

        month.addActionListener((e)->{
            dialogMonthSelection = new
DialogMonthSelection(window, ContractWindow.this, "Получение
месяца");
            dialogMonthSelection.setVisible(true);
            if (monthDate != null)
            {
                String msg = " Заключенные договоры за этот месяц:
" + contractService.findNew(monthDate) +
```

```
                        "\n Выполненные договоры за этот месяц: "
+ contractService.findFinish(monthDate) +
                        "\n Прибыль за этот месяц: " +
contractService.findFinishIncome(monthDate);
                JOptionPane.showMessageDialog(null, msg);
            }
        });

        month.setMnemonic(KeyEvent.VK_D);


        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataContracts.setRowSorter(sorter);
            }
        });


        outdated.addActionListener((e) -> {
            log.info("Старт outdated listener");
            if (dataContracts.getRowCount() > 0)
            {
                try
                {
                    List<Contract> contractList =
contractService.findOutdated();
                    String [][] d = new
String[contractList.size()][5];
                    for (int i = 0; i < contractList.size(); i++)
                    {
                        d[i] =
contractList.get(i).toTableFormat();
                    }
```

```
                    model.setDataVector(d, columns);
                    model.fireTableDataChanged();
                }
                catch (Exception ex)
                {
                    JOptionPane.showMessageDialog(null, "Ошибка");
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });

        outdated.setMnemonic(KeyEvent.VK_D);

        time.addActionListener((e) -> {
            log.info("Старт time listener");
            if (dataContracts.getRowCount() > 0)
            {
                try
                {
                    dialogTimePeriodSelection = new
DialogTimePeriodSelection(window, ContractWindow.this, "Установка
периода времени");
                    dialogTimePeriodSelection.setVisible(true);

                    if (dateBegin != null && dateEnd != null)
                    {
                        if (dateBegin.getTime() >
dateEnd.getTime())
                            throw new UnacceptableTimePeriod();

                        List<Contract> contractList =
contractService.findTimePeriod(dateBegin, dateEnd);
                        String[][] d = new
String[contractList.size()][5];
                        for (int i = 0; i < contractList.size();
i++) {
                            d[i] =
contractList.get(i).toTableFormat();
                        }

                        model.setDataVector(d, columns);
                        model.fireTableDataChanged();
                    }
                }
                catch (Exception ex)
                {
```

```
                    JOptionPane.showMessageDialog(null, "Ошибка:"
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });

        time.setMnemonic(KeyEvent.VK_D);

        drop.addActionListener((e) -> {
            log.info("Старт drop listener");
            if (dataContracts.getRowCount() > 0)
            {
                try
                {
                    List<Contract> contractList =
contractService.findAll();
                    String [][] d = new
String[contractList.size()][5];
                    for (int i = 0; i < contractList.size(); i++)
                    {
                        d[i] =
contractList.get(i).toTableFormat();
                    }

                    model.setDataVector(d, columns);
                    model.fireTableDataChanged();
                }
                catch (Exception ex)
                {
                    JOptionPane.showMessageDialog(null, "Ошибка");
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });

        drop.setMnemonic(KeyEvent.VK_D);


        window.setVisible(true);
    }
```

```
    /**
     * Метод проверки списка на отсутсвие записей
     * @throws EmptyFileException моё исключение
     */
    private void checkList() throws EmptyFileException
    {
        if(model.getRowCount() == 0)
            throw new EmptyFileException();
    }

    /** Метод загрузки данных в XML файл */
    public void makeXml() {
        try {
            // Создание парсера документа
            DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
            // Создание пустого документа
            Document doc = builder.newDocument();
            // Создание корневого элемента window и добавление его
в документ
            Node window = doc.createElement("window");
            doc.appendChild(window);
            // Создание дочерних элементов dataEmploy и присвоение
значений атрибутам
            for (int i = 0; i < model.getRowCount(); i++) {
                Element dataManager =
doc.createElement("dataContracts");
                window.appendChild(dataManager);
                dataManager.setAttribute("description", (String)
model.getValueAt(i, 1));
                dataManager.setAttribute("price", (String)
model.getValueAt(i, 2));
                dataManager.setAttribute("client", (String)
model.getValueAt(i, 3));
                dataManager.setAttribute("manager", (String)
model.getValueAt(i, 4));
                dataManager.setAttribute("dateBegin", (String)
model.getValueAt(i, 5));
                dataManager.setAttribute("dateEnd", (String)
model.getValueAt(i, 6));
                dataManager.setAttribute("isEnd", (String)
model.getValueAt(i, 7));
            }
            try {
                // Создание преобразователя документа
                Transformer trans =
TransformerFactory.newInstance().newTransformer();
                // Создание файла с именем dataEmploy.xml для
записи документа
                java.io.FileWriter fw = new
FileWriter("dataContracts.xml");
```

```
                // Запись документа в файл
                trans.transform(new DOMSource(doc), new
StreamResult(fw));
            } catch (TransformerConfigurationException e) {
                e.printStackTrace();
            } catch (TransformerException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }


        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param arr - данные, полученные от пользователя
     */
    public void addR(String[] arr, Date begin, Date end)
    {
        String[] r = arr[2].split("\u2588");
        int client_id = Integer.parseInt(r[0]);
        r = arr[3].split("\u2588");
        int manager_id = Integer.parseInt(r[0]);

        Contract newM = new Contract(arr[0],
                Double.parseDouble(arr[1]),
                clientService.findById(client_id),
                managerService.findById(manager_id),
                null,
                begin,
                end,
                false);
        contractService.persist(newM);
        model.addRow(newM.toTableFormat());
    }

    /**
     * Вспомогательный метод изменения данных в таблице
     * @param arr - данные, полученные от пользователя
     */
    public void editR(String[] arr, Date begin, Date end)
    {
        Contract C =
contractService.findById(Integer.parseInt(arr[0]));
        C.setDescription(arr[1]);
        C.setPrice(Double.parseDouble(arr[2]));
        String[] r = arr[3].split("\u2588");
```

```
            int client_id = Integer.parseInt(r[0]);
            C.setClient(clientService.findById(client_id));
            r = arr[4].split(" ");
            int manager_id = Integer.parseInt(r[0]);
            C.setManager(managerService.findById(manager_id));
            C.setDateBegin(begin);
            C.setDateEnd(end);
            contractService.update(C);
        }

        /**
         * Вспомогательный метод получения даты для поиска
         * @param begin - начало временного отрезка
         * @param end - конец временного отрезка
         */
        public void setDate(Date begin, Date end)
        {
            dateBegin = begin;
            dateEnd = end;
        }

        /**
         * Вспомогательный метод получения месяца для отчета
         * @param date - дата с месяцем
         */
        public void setMonth(Date date)
        {
            monthDate = date;
        }

        public String[] getManagers()
        {
            List<Manager> managers = managerService.findAll();
            String [] result = new String[managers.size()];
            for (int i = 0; i < managers.size(); i++)
                result[i] = managers.get(i).getId() + " " +
managers.get(i).getName() + " " + managers.get(i).getSurname();
            return result;
        }

        public String[] getClients()
        {
            List<Client> clients = clientService.findAll();
            String [] result = new String[clients.size()];
            for (int i = 0; i < clients.size(); i++)
                result[i] = clients.get(i).getId() + " " +
clients.get(i).getName() + " " + clients.get(i).getSurname();
            return result;
        }

    }
```

WorkerContractWindow.java

```
package Factory.gui;

import Factory.model.*;
import Factory.service.*;
import Factory.exceptions.*;
import Factory.util.ReportUtil;

import org.w3c.dom.*;

import javax.swing.filechooser.FileFilter;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.io.*;
import java.util.List;
import java.util.Date;
import java.util.logging.*;

/** Класс приложения, визуализирующий экранную форму с договорами,
которые выполняет рабочий */
public class WorkerContractWindow
{
    /** Стандартный конструктор */
    WorkerContractWindow(int workerId)
    {
        this.workerId = workerId;
        show();
    }

    /** Идентификатор рабочего, для которого открыто окно */
    private int workerId;

    /** Окно приложения */
    private JFrame window;

    /** Модель таблицы */
```

```
    private DefaultTableModel model;

    /** Добавить */
    private JButton add;

    /** Удалить */
    private JButton delete;

    /** Печать */
    private JButton print;

    /** Показать описание */
    private JButton description;

    /** Панель инструментов */
    private JToolBar toolBar;

    /** Таблица */
    protected JTable dataContracts;

    /** Поле поискового запроса */
    private JTextField textSearch;

    /** Поиск */
    private JButton search;

    /** Показать просроченные договоры */
    private JButton outdated;

    /** Показать договоры за указанный период */
    private JButton time;

    /** Сбросить */
    private JButton drop;

    /** Скролл */
    private JScrollPane scroll;

    /** Сервис рабочих */
    private EmployeeService workerService = new EmployeeService();

    /** Сервис контрактов */
    private ContractService contractService = new
ContractService();

    /** Поток 2 отвечает за формирование отчет */
    Thread t2 = new Thread();

    /** Логгер класса */
```

```
    private static final Logger log =
Logger.getLogger(WorkerContractWindow.class.getName());

    /** Диалог добавления данных */
    private AddDialogWorkerContract addDialog;

    /** Диалог изменения временного промежутка */
    private DialogTimePeriodSelection dialogTimePeriodSelection;

    /** Начало временного отрезка для сортировки */
    private Date dateBegin;

    /** Конец временного отрезка для сортировки */
    private Date dateEnd;

    public void show()
    {
        log.info("Открытие окна WorkerContractWindow");
        window = new JFrame("factory: Список договоров завода");
        window.setSize(1000,500);
        window.setLocation(310,130);
        window.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        // Создание кнопок и прикрепление иконок
        log.info("Добавление кнопок к окну WorkerContractWindow");
        add = new JButton("Добавить");
        delete = new JButton("Удалить");
        print = new JButton("Печать");
        description = new JButton("Описание");

        // Настройка подсказок
        add.setToolTipText("Добавить контракт");
        delete.setToolTipText("Удалить контракт");
        print.setToolTipText("Распечатать контракты");
        description.setToolTipText("Показать описание контракта");
        // Добавление кнопок на панель инструментов
        toolBar = new JToolBar("Панель инструментов");
        toolBar.add(add);
        toolBar.add(delete);
        toolBar.add(print);
        toolBar.add(description);
        // Размещение панели инструментов
        window.setLayout(new BorderLayout());
        window.add(toolBar,BorderLayout.NORTH);

        // Создание таблицы с данными
        String[] columns = {"ID", "Описание", "Цена", "Клиент",
"Менеджер", "Дата подписания", "Дата окончания работ",
"Состояние"};
        log.info("Добавление таблицы с данными к окну
ContractWindow");
```

```
        List<Contract> contractsList =
(workerService.findById(workerId)).getContracts();
        String [][] data = new String[contractsList.size()][5];
        for (int i = 0; i < contractsList.size(); i++)
        {
            data[i] = contractsList.get(i).toTableFormat();
        }

        // Настройка таблицы
        model = new DefaultTableModel(data, columns)
        {
            public boolean isCellEditable(int rowIndex, int
columnIndex)
            {
                return false;
            }
        };
        this.dataContracts = new JTable(model);
        RowSorter<TableModel> sort = new
TableRowSorter<TableModel>(model);
        dataContracts.setRowSorter(sort);
        dataContracts.setFont(new Font(Font.SERIF,Font.BOLD,14));
        dataContracts.setIntercellSpacing(new Dimension(0,1));
        dataContracts.setRowHeight(dataContracts.getRowHeight()
+10);

dataContracts.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);


dataContracts.setDefaultRenderer(dataContracts.getColumnClass(1),
new DefaultTableCellRenderer(){
            public Component getTableCellRendererComponent(JTable
table, Object value, boolean isSelected, boolean hasFocus, int
row, int column) {

super.setHorizontalAlignment(SwingConstants.CENTER);
                super.getTableCellRendererComponent(table, value,
isSelected, hasFocus, row, column);
                return this;
            }


        });

        // Если не выделена строка, то прячем кнопки

dataContracts.getSelectionModel().addListSelectionListener((e) ->
{
            boolean check = !
dataContracts.getSelectionModel().isSelectionEmpty();
            delete.setVisible(check);
            description.setVisible(check);
```

```
            print.setVisible(check);
        });

        scroll = new JScrollPane(this.dataContracts);

        // Размещение таблицы с данными
        window.add(scroll,BorderLayout.CENTER);

        // Подготовка компонентов поиска
        textSearch = new JTextField();
        textSearch.setColumns(20);
        search = new JButton("Поиск");
        outdated = new JButton("Просроченные");
        time = new JButton("Период");
        drop = new JButton("Сброс");
        outdated.setToolTipText("Показать просроченные договоры");
        time.setToolTipText("Показать договоры за определенный
промежуток времени");
        window.getRootPane().setDefaultButton(search);
        // remove the binding for pressed

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
        // retarget the binding for released

window.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW
)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
        // Добавление компонентов на панель
        JPanel searchPanel = new JPanel();
        searchPanel.add(textSearch);
        searchPanel.add(search);
        searchPanel.add(outdated);
        searchPanel.add(time);
        searchPanel.add(drop);

        // Размещение панели поиска внизу окна
        window.add(searchPanel,BorderLayout.SOUTH);

        add.addActionListener((e) -> {
            log.info("Старт Add listener");
            addDialog = new AddDialogWorkerContract(window,
WorkerContractWindow.this, "Добавление записи");
            addDialog.setVisible(true);
        });

        add.setMnemonic(KeyEvent.VK_A);
        delete.addActionListener((e) -> {
            log.info("Старт Delete listener");
```

```
            if (dataContracts.getRowCount() > 0) {
                if (dataContracts.getSelectedRow() != -1) {
                    try
                    {
                        Contract c =
contractService.findById(Integer.parseInt(dataContracts.getValueAt
(dataContracts.getSelectedRow(), 0).toString()));

(workerService.findById(workerId)).removeContract(c);

model.removeRow(dataContracts.convertRowIndexToModel(dataContracts
.getSelectedRow()));
                        contractService.update(c);
                        JOptionPane.showMessageDialog(window, "Вы
удалили строку");
                        log.info("Была удалена строка данных");
                    }
                    catch (Exception ex)
                    {
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали строку для удаления");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка для удаление");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей. Нечего удалять");
                log.log(Level.WARNING, "Исключение: нет записей.
нечего удалять");
            }
        });

        delete.setMnemonic(KeyEvent.VK_D);

        description.addActionListener((e) -> {
            log.info("Старт Description listener");
            if (dataContracts.getRowCount() > 0) {
                if (dataContracts.getSelectedRow() != -1) {
                    try
                    {
                        JOptionPane.showMessageDialog(window,
dataContracts.getValueAt(dataContracts.getSelectedRow(),
1).toString());
                    }
                    catch (Exception ex)
                    {
```

```
                        JOptionPane.showMessageDialog(null,
"Ошибка");
                        log.log(Level.SEVERE, "Исключение: ", ex);
                    }
                } else {
                    JOptionPane.showMessageDialog(null, "Вы не
выбрали договор");
                    log.log(Level.WARNING, "Исключение: не выбрана
строка");
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });

        description.setMnemonic(KeyEvent.VK_D);

        print.addActionListener((e)->{
            log.info("Старт Print listener");
            t2 = new Thread(() -> {
                try {
                    JFileChooser fileChooser = new JFileChooser();
                    fileChooser.setDialogTitle("Select where you
want to save the report");
                    FileFilter pdf = new
FileNameExtensionFilter("PDF file(.pdf)", "pdf");
                    fileChooser.addChoosableFileFilter(pdf);
                    fileChooser.setCurrentDirectory(new
File("."));
                    String resultpath = "reportContracts.pdf";
                    int returnVal =
fileChooser.showSaveDialog(null);
                    if(returnVal == JFileChooser.APPROVE_OPTION)
                    {
                        File file = fileChooser.getSelectedFile();
                        resultpath = file.getAbsolutePath();
                    }
                    checkList();
                    makeXml();
                    ReportUtil.print("dataContracts.xml",
"window/dataContracts", "contracts.jrxml", resultpath);
                    JOptionPane.showMessageDialog(null, "2 поток
закончил работу. Отчет создан");
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(null, "Ошибка: "
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            });
```

```
            t2.start();
        });

        search.addActionListener((e) -> {
            if (model.getRowCount() != 0) {
                if (!textSearch.getText().isEmpty())
                    log.info("Запуск нового поиска по ключевому
слову: " + textSearch.getText());
                else
                    log.info("Сброс ключевого слова поиска");
                TableRowSorter<TableModel> sorter = new
TableRowSorter<TableModel>(((DefaultTableModel) model));
                sorter.setStringConverter(new
TableStringConverter() {
                    @Override
                    public String toString(TableModel model, int
row, int column) {
                        return model.getValueAt(row,
column).toString().toLowerCase();
                    }
                });
                sorter.setRowFilter(RowFilter.regexFilter("(?i)" +
textSearch.getText().toLowerCase()));
                dataContracts.setRowSorter(sorter);
            }
        });

        outdated.addActionListener((e) -> {
            log.info("Старт outdated listener");
            if (dataContracts.getRowCount() > 0)
            {
                try
                {
                    List<Contract> contractList =
contractService.findOutdated(workerId);
                    String [][] d = new
String[contractList.size()][5];
                    for (int i = 0; i < contractList.size(); i++)
                    {
                        d[i] =
contractList.get(i).toTableFormat();
                    }

                    model.setDataVector(d, columns);
                    model.fireTableDataChanged();
                }
                catch (Exception ex)
                {
                    JOptionPane.showMessageDialog(null, "Ошибка");
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
```

```
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });

        outdated.setMnemonic(KeyEvent.VK_D);

        time.addActionListener((e) -> {
            log.info("Старт time listener");
            if (dataContracts.getRowCount() > 0)
            {
                try
                {
                    dialogTimePeriodSelection = new
DialogTimePeriodSelection(window, WorkerContractWindow.this,
"Установка периода времени");
                    dialogTimePeriodSelection.setVisible(true);

                    if (dateBegin != null && dateEnd != null)
                    {
                        if (dateBegin.getTime() >
dateEnd.getTime())
                            throw new UnacceptableTimePeriod();

                        List<Contract> contractList =
contractService.findTimePeriod(dateBegin, dateEnd, workerId);
                        String[][] d = new
String[contractList.size()][5];
                        for (int i = 0; i < contractList.size();
i++) {
                            d[i] =
contractList.get(i).toTableFormat();
                        }

                        model.setDataVector(d, columns);
                        model.fireTableDataChanged();
                    }
                }
                catch (Exception ex)
                {
                    JOptionPane.showMessageDialog(null, "Ошибка:"
+ ex.toString());
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
```

```
        });

        time.setMnemonic(KeyEvent.VK_D);

        drop.addActionListener((e) -> {
            log.info("Старт drop listener");
            if (dataContracts.getRowCount() > 0)
            {
                try
                {
                    List<Contract> contractList =
(workerService.findById(workerId)).getContracts();
                    String [][] d = new
String[contractList.size()][5];
                    for (int i = 0; i < contractList.size(); i++)
                    {
                        d[i] =
contractList.get(i).toTableFormat();
                    }

                    model.setDataVector(d, columns);
                    model.fireTableDataChanged();
                }
                catch (Exception ex)
                {
                    JOptionPane.showMessageDialog(null, "Ошибка");
                    log.log(Level.SEVERE, "Исключение: ", ex);
                }
            } else {
                JOptionPane.showMessageDialog(null, "В данном окне
нет записей");
                log.log(Level.WARNING, "Исключение: нет записей");
            }
        });

        drop.setMnemonic(KeyEvent.VK_D);

        window.setVisible(true);
    }

    /**
     * Метод проверки списка на отсутсвие записей
     * @throws EmptyFileException моё исключение
     */
    private void checkList() throws EmptyFileException
    {
        if(model.getRowCount() == 0)
            throw new EmptyFileException();
    }
```

```
    /** Метод загрузки данных в XML файл */
    public void makeXml() {
        try {
            // Создание парсера документа
            DocumentBuilder builder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
            // Создание пустого документа
            Document doc = builder.newDocument();
            // Создание корневого элемента window и добавление его
в документ
            Node window = doc.createElement("window");
            doc.appendChild(window);
            // Создание дочерних элементов dataEmploy и присвоение
значений атрибутам
            for (int i = 0; i < model.getRowCount(); i++) {
                Element dataManager =
doc.createElement("dataContracts");
                window.appendChild(dataManager);
                dataManager.setAttribute("description", (String)
model.getValueAt(i, 1));
                dataManager.setAttribute("price", (String)
model.getValueAt(i, 2));
                dataManager.setAttribute("client", (String)
model.getValueAt(i, 3));
                dataManager.setAttribute("manager", (String)
model.getValueAt(i, 4));
                dataManager.setAttribute("dateBegin", (String)
model.getValueAt(i, 5));
                dataManager.setAttribute("dateEnd", (String)
model.getValueAt(i, 6));
                dataManager.setAttribute("isEnd", (String)
model.getValueAt(i, 7));
            }
            try {
                // Создание преобразователя документа
                Transformer trans =
TransformerFactory.newInstance().newTransformer();
                // Создание файла с именем dataEmploy.xml для
записи документа
                java.io.FileWriter fw = new
FileWriter("dataClients.xml");
                // Запись документа в файл
                trans.transform(new DOMSource(doc), new
StreamResult(fw));
            } catch (TransformerConfigurationException e) {
                e.printStackTrace();
            } catch (TransformerException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        }
    }

    /**
     * Вспомогательный метод добавления данных в таблицу
     * @param ID - данные, полученные от пользователя
     */
    public void addR(int ID)
    {
        Contract c = contractService.findById(ID);
        (workerService.findById(workerId)).addContract(c);
        model.addRow(c.toTableFormat());
        contractService.update(c);
    }

    /**
     * Вспомогательный метод получения даты для поиска
     * @param begin - начало временного отрезка
     * @param end - конец временного отрезка
     */
    public void setDate(Date begin, Date end)
    {
        dateBegin = begin;
        dateEnd = end;
    }


}
```

DialogSpecialisation.java

```
package Factory.gui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Абстрактный класс диалогового окна Добавления/Редактирования
данных профессии
 */
public abstract class DialogSpecialisation extends JDialog {

    /** Текстовое поле названия */
```

```
    protected JTextField name;

    /** Переменная корректности ввода */
    protected boolean check = false;

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");

    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");

    /**
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public abstract void progress(SpecialisationWindow parent);

    /**
     * Инициализация
     *
     * @param parent - Объект класса приложения
     */
    public abstract void init(SpecialisationWindow parent);

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogSpecialisation(JFrame owner, SpecialisationWindow
parent, String title) {
        super(owner, title, true);
        ok.setEnabled(false);
        // Инит кнопок
        init(parent);

        // Создание валидатора полей
        name.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(name);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(name);
```

```
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        JPanel mainp = new JPanel();

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2, 2, 2));

        panel.setSize(300, 100);

        // adds to the GridLayout
        panel.add(new JLabel("Название профессии"));
        panel.add(name);
        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }

    /**
     * Проверка поля на корректность введенных данных
     * @param field - проверяемое поле
     */
    protected void checker(JTextField field)
    {
        Pattern r = Pattern.compile("^[A-ЯЁ][a-яЁё]{1,10}$");
        Matcher m = r.matcher(field.getText());
        if (m.matches()) {
```

```
field.setBorder(BorderFactory.createLineBorder(Color.GREEN));
            check = true;
        }
        else {

field.setBorder(BorderFactory.createLineBorder(Color.RED));
            check = false;
        }

        ok.setEnabled(check);
    }
}
```

DialogWorker.java

```
package Factory.gui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Абстрактный класс диалогового окна Добавления/Редактирования
данных сотрудников
 */
public abstract class DialogWorker extends JDialog {

    /** Текстовое поле имени */
    protected JTextField name;

    /** Текстовое поле фамилии */
    protected JTextField surname;

    /** Текстовое поле опыта работы */
    protected JTextField exp;

    /** Выпадающий список профессий */
    protected JComboBox specs;

    /** Массив переменных, отвечающих за корректность ввода */
    protected Boolean[] check = {false, false, false};

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");

    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");

    /**
```

```
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public abstract void progress(WorkerWindow parent);

    /**
     * Инициализация
     *
     * @param parent - Объект класса приложения
     */
    public abstract void init(WorkerWindow parent);

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogWorker(JFrame owner, WorkerWindow parent, String
title) {
        super(owner, title, true);
        ok.setEnabled(false);
        // Инит кнопок
        init(parent);

        // Создание валидатора полей
        name.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(0, name);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(0, name);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        surname.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
```

```
                checker(1,surname);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(1,surname);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        exp.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checkerInt(2,exp);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checkerInt(2,exp);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });


        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        JPanel mainp = new JPanel();

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(4, 2, 2, 2));

        panel.setSize(300, 300);

        // adds to the GridLayout
        panel.add(new JLabel("Имя"));
        panel.add(name);
        panel.add(new JLabel("Фамилия"));
        panel.add(surname);
        panel.add(new JLabel("Опыт работы"));
        panel.add(exp);
        panel.add(new JLabel("Специализация"));
```

```
        panel.add(specs);
        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }

    /**
     * Проверка поля на корректность введенных данных
     * Проверяет поля, которые ддолжны быть строками
     *
     * @param i - номер поля
     * @param field - проверяемое поле
     */
    protected void checker(int i, JTextField field){
        Pattern b = Pattern.compile("^[A-ЯЁ][а-яЁё]{1,10}$");
        Matcher rr = b.matcher(field.getText());
        Color color = Color.GREEN;
        if (rr.matches()) {
            check[i] = true;
        } else {
            color = Color.RED;
            check[i] = false;
        }
        field.setBorder(BorderFactory.createLineBorder(color));

        ok.setEnabled(check[0] && check[1] && check[2]);
    }

    /**
     * Проверка поля на корректность введенных данных
     * Проверяет поля, которые ддолжны быть целыми числами
     *
     * @param i - номер поля
     * @param field - проверяемое поле
     */
    protected void checkerInt(int i, JTextField field){
```

```
            Pattern c = Pattern.compile("^[+-]?(([1-9][0-9]*)|(0))$");
            Matcher rm = c.matcher(field.getText());
            Color color = Color.GREEN;
            if (rm.matches()) {
                check[i] = true;
            } else {
                color = Color.RED;
                check[i] = false;
            }
            field.setBorder(BorderFactory.createLineBorder(color));

            ok.setEnabled(check[0] && check[1] && check[2]);
        }
    }
```

DialogManager.java

```
package Factory.gui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Абстрактный класс диалогового окна Добавления/Редактирования
данных менеджеров
 */
public abstract class DialogManager extends JDialog {

    /** Текстовое поле имени */
    protected JTextField name;

    /** Текстовое поле фамилии */
    protected JTextField surname;

    /** Массив переменных, отвечающих за корректность ввода */
    protected Boolean[] check = {false,false};

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");

    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");


    /**
     * Выполнение манипуляций с данными
     *
```

```
     * @param parent - Объект класса приложения
     */
    public abstract void progress(ManagerWindow parent);

    /**
     * Инициализация
     *
     * @param parent - Объект класса приложения
     */
    public abstract void init(ManagerWindow parent);

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogManager(JFrame owner, ManagerWindow parent,
String title) {
        super(owner, title, true);
        ok.setEnabled(false);
        // Инит кнопок
        init(parent);

        // Создание валидатора полей
        name.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(0, name);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(0, name);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        surname.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(1,surname);
            }
```

```
            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(1,surname);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        JPanel mainp = new JPanel();

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2, 2, 2));

        panel.setSize(300, 100);
        // adds to the GridLayout
        panel.add(new JLabel("Имя"));
        panel.add(name);
        panel.add(new JLabel("Фамилия"));
        panel.add(surname);
        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }

    /**
     * Проверка поля на корректность введенных данных
     *
     * @param i - номер поля
     * @param field - проверяемое поле
```

```
     */
    protected void checker(int i, JTextField field)
    {
        Pattern r = Pattern.compile("^[А-ЯЁ][а-яЁё]{1,10}$");
        Matcher m = r.matcher(field.getText());
        if (m.matches()) {

field.setBorder(BorderFactory.createLineBorder(Color.GREEN));
            check[i] = true;
        }
        else {

field.setBorder(BorderFactory.createLineBorder(Color.RED));
            check[i] = false;
        }

        ok.setEnabled(check[0] && check[1]);
    }
}
```

DialogClient.java

```
package Factory.gui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Абстрактный класс диалогового окна Добавления/Редактирования
данных клиентов
 */
public abstract class DialogClient extends JDialog {

    /** Текстовое поле имени */
    protected JTextField name;

    /** Текстовое поле фамилии */
    protected JTextField surname;

    /** Текстовое поле компании */
    protected JTextField company;

    /** Массив переменных, отвечающих за корректность ввода */
    protected Boolean[] check = {false, false, false};

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");
```

```
    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");

    /**
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public abstract void progress(ClientWindow parent);

    /**
     * Инициализация
     *
     * @param parent - Объект класса приложения
     */
    public abstract void init(ClientWindow parent);

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogClient(JFrame owner, ClientWindow parent, String
title) {
        super(owner, title, true);
        ok.setEnabled(false);
        // Инит кнопок
        init(parent);

        // Создание валидатора полей
        name.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(0, name);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(0, name);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });
```

```
        surname.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(1,surname);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(1,surname);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        company.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(2,company);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(2,company);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        JPanel mainp = new JPanel();

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2, 2, 2));

        panel.setSize(300, 100);

        // adds to the GridLayout
        panel.add(new JLabel("Имя"));
        panel.add(name);
        panel.add(new JLabel("Фамилия"));
        panel.add(surname);
```

```
        panel.add(new JLabel("Компания"));
        panel.add(company);
        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }

    /**
     * Проверка поля на корректность введенных данных
     *
     * @param i - номер поля
     * @param field - проверяемое поле
     */
    protected void checker(int i, JTextField field)
    {
        Pattern r = Pattern.compile("^[A-ЯЁ][a-яЁё]{1,10}$");
        Matcher m = r.matcher(field.getText());
        if (m.matches()) {

field.setBorder(BorderFactory.createLineBorder(Color.GREEN));
            check[i] = true;
        }
        else {

field.setBorder(BorderFactory.createLineBorder(Color.RED));
            check[i] = false;
        }

        ok.setEnabled(check[0] && check[1] && check[2]);
    }
}
```

DialogClient.java

```
package Factory.gui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
```

```java
import javax.swing.event.DocumentListener;
import org.jdatepicker.impl.*;
import java.awt.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Абстрактный класс диалогового окна Добавления/Редактирования
данных менеджеров
 */
public abstract class DialogContract extends JDialog
{

    /** Текстовое поле описания */
    protected JTextField description;

    /** Текстовое поле цены */
    protected JTextField price;

    /** Выпадающий список клиентов */
    protected JComboBox clients;

    /** Выпадающий список менеджеров */
    protected JComboBox managers;

    /** Дата пикер даты подписания контракта */
    protected JDatePickerImpl dataBegin;

    /** Дата пикер даты окончания контракта */
    protected JDatePickerImpl dataEnd;

    /** Переменная корректности ввода */
    protected boolean check = false;

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");

    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");

    /**
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public abstract void progress(ContractWindow parent);

    /**
     * Инициализация
     *
```

```
     * @param parent - Объект класса приложения
     */
    public abstract void init(ContractWindow parent);

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogContract(JFrame owner, ContractWindow parent,
String title)
    {
        super(owner, title, true);

        JLabel descriptionLab = new JLabel("Описание");
        JLabel priceLab = new JLabel("Цена");
        JLabel clientLab = new JLabel("Клиент");
        JLabel managerLab = new JLabel("Менеджер");
        JLabel beginLab = new JLabel("Дата подписания договора");
        JLabel endLab = new JLabel("Дата окончания работ");

        ok.setEnabled(false);
        // Инит кнопок
        init(parent);


        price.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checkerFloat(price);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checkerFloat(price);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        JPanel mainp = new JPanel();
```

```
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(6, 2, 2, 2));

        panel.setSize(300, 300);

        // adds to the GridLayout
        panel.add(descriptionLab);
        panel.add(description);
        panel.add(priceLab);
        panel.add(price);
        panel.add(clientLab);
        panel.add(clients);
        panel.add(managerLab);
        panel.add(managers);
        panel.add(beginLab);
        panel.add(dataBegin);
        panel.add(endLab);
        panel.add(dataEnd);

        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 250);
        this.getRootPane().setDefaultButton(ok);
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }

    /**
     * Проверка поля на корректность введенных данных
     *
     * @param field - проверяемое поле
     */
    protected void checkerFloat(JTextField field)
    {
        Pattern r = Pattern.compile("^[+-]?(([1-9][0-9]*)|(0))
([.,][0-9]+)?$");
        Matcher m = r.matcher(field.getText());
        if (m.matches()) {
```

```
field.setBorder(BorderFactory.createLineBorder(Color.GREEN));
            check = true;
        }
        else {

field.setBorder(BorderFactory.createLineBorder(Color.RED));
            check = false;
        }

        ok.setEnabled(check);
    }


}
```

DialogMonthSelection.java

```
package Factory.gui;

import Factory.util.DateLabelFormatter;
import org.jdatepicker.impl.JDatePanelImpl;
import org.jdatepicker.impl.JDatePickerImpl;
import org.jdatepicker.impl.UtilDateModel;

import javax.swing.*;
import java.awt.*;
import java.util.Date;
import java.util.Properties;

/**
 * Класс диалогового окна получения месяца для отчета по заводу
 */
public class DialogMonthSelection extends JDialog {

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");

    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");

    /** Дата пикер месяца */
    protected JDatePickerImpl month;

    /**
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public void progress(ContractWindow parent)
    {
        setVisible(false);
        parent.setMonth((Date) month.getModel().getValue());
```

```
    }

    /**
     * Инициализация
     */
    public void init()
    {
        UtilDateModel model = new UtilDateModel();
        Properties p = new Properties();
        p.put("text.today", "Today");
        p.put("text.month", "Month");
        p.put("text.year", "Year");
        JDatePanelImpl datePanel = new JDatePanelImpl(model, p);
        month = new JDatePickerImpl(datePanel, new
DateLabelFormatter());
    }

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogMonthSelection(JFrame owner, ContractWindow
parent, String title)
    {
        super(owner, title, true);
        // Инит кнопок
        init();
        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());

        JPanel mainp = new JPanel();
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(1, 2, 2, 2));

        panel.setSize(300, 100);

        // adds to the GridLayout
        panel.add(new Label("Выберите месяц (день не важен)"));
        panel.add(month);
        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);
```

```
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }
}
```

DialogTimePeriodSelection.java

```
package Factory.gui;

import Factory.util.DateLabelFormatter;
import org.jdatepicker.impl.JDatePanelImpl;
import org.jdatepicker.impl.JDatePickerImpl;
import org.jdatepicker.impl.UtilDateModel;

import javax.swing.*;
import java.awt.*;
import java.util.Date;
import java.util.Properties;

/**
 * Класс Диалогового окна получения временного периода для
отображения данных договоров
 */
public class DialogTimePeriodSelection extends JDialog {

    /** Кнопка принять */
    private JButton ok = new JButton("Принять");

    /** Кнопка отменить */
    private JButton cancel = new JButton("Закрыть");

    /** Дата пикер начала временного отрезка */
    protected JDatePickerImpl dataBegin;

    /** Дата пикер конца временного отрезка */
    protected JDatePickerImpl dataEnd;

    /**
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public void progress(ContractWindow parent)
    {
        setVisible(false);
```

```
        parent.setDate((Date) dataBegin.getModel().getValue(),
(Date) dataEnd.getModel().getValue());
    }
    public void progress(WorkerContractWindow parent)
    {
        setVisible(false);
        parent.setDate((Date) dataBegin.getModel().getValue(),
(Date) dataEnd.getModel().getValue());
    }

    /**
     * Инициализация
     */
    public void init()
    {
        UtilDateModel model1 = new UtilDateModel();
        UtilDateModel model2 = new UtilDateModel();
        Properties p = new Properties();
        p.put("text.today", "Today");
        p.put("text.month", "Month");
        p.put("text.year", "Year");
        JDatePanelImpl datePanel1 = new JDatePanelImpl(model1, p);
        JDatePanelImpl datePanel2 = new JDatePanelImpl(model2, p);
        dataBegin = new JDatePickerImpl(datePanel1, new
DateLabelFormatter());
        dataEnd = new JDatePickerImpl(datePanel2, new
DateLabelFormatter());
    }

    /**
     * Показать интерфейс
     */
    public void display()
    {
        JPanel mainp = new JPanel();
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(2, 2, 2, 2));

        panel.setSize(300, 100);

        // adds to the GridLayout
        panel.add(new Label("От"));
        panel.add(dataBegin);
        panel.add(new Label("До"));
        panel.add(dataEnd);
        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
```

```
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);
// remove the binding for pressed

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");
// retarget the binding for released

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
    }

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public DialogTimePeriodSelection(JFrame owner, ContractWindow
parent, String title)
    {
        super(owner, title, true);
        // Инит кнопок
        init();
        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        display();
    }
    public DialogTimePeriodSelection(JFrame owner,
WorkerContractWindow parent, String title)
    {
        super(owner, title, true);
        // Инит кнопок
        init();
        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        display();
    }

}
```

AddDialogSpecialisation.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна добавления данных профессии
```

```
 */
public class AddDialogSpecialisation extends DialogSpecialisation
{

    @Override
    public void progress(SpecialisationWindow parent) {
        setVisible(false);

        parent.addR(name.getText());

        JOptionPane.showMessageDialog(null, "Вы добавили профессию
\""+name.getText()+"\"");
    }

    @Override
    public void init(SpecialisationWindow parent)
    {
        name = new JTextField(20);
    }

    /** Конструктор окна добавления данных профессии
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public AddDialogSpecialisation(JFrame owner,
SpecialisationWindow parent, String title)
    {
        super(owner,parent,title);
    }
}
```

AddDialogWorker.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна добавления данных рабочего
 */
public class AddDialogWorker extends DialogWorker
{
    @Override
    public void progress(WorkerWindow parent) {
        setVisible(false);
        String[] arr = {name.getText(), surname.getText(),
exp.getText(), (String) specs.getSelectedItem()};

        parent.addR(arr);
```

```
        JOptionPane.showMessageDialog(null, "Вы добавили
сотрудника \""+arr[0]+" "+arr[1]+"\"");
    }

    @Override
    public void init(WorkerWindow parent)
    {
        name = new JTextField(20);
        surname = new JTextField(20);
        exp = new JTextField(20);
        specs = new JComboBox(parent.getSpecs());
    }

    /** Конструктор окна добавления данных рабочего
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public AddDialogWorker(JFrame owner, WorkerWindow parent,
String title)
    {
        super(owner, parent,title);
    }
}
```

AddDialogManager.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна добавления данных менеджера
 */
public class AddDialogManager extends DialogManager {

    @Override
    public void progress(ManagerWindow parent) {
        setVisible(false);
        String[] arr = {name.getText(),surname.getText()};

        parent.addR(arr);

        JOptionPane.showMessageDialog(null, "Вы добавили
сотрудника \""+arr[0]+" "+arr[1]+"\"");
    }

    @Override
    public void init(ManagerWindow parent)
```

```
    {
        name = new JTextField(20);
        surname= new JTextField(20);
    }

    /** Конструктор окна добавления данных менеджера
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public AddDialogManager(JFrame owner, ManagerWindow parent,
String title)
    {
        super(owner,parent,title);
    }
}
```

AddDialogClient.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна добавления данных клиента
 */
public class AddDialogClient extends DialogClient {

    @Override
    public void progress(ClientWindow parent) {
        setVisible(false);
        String[] arr =
{name.getText(),surname.getText(),company.getText()};

        parent.addR(arr);

        JOptionPane.showMessageDialog(null, "Вы добавили клиента
\""+arr[0]+" "+arr[1]+"\"");
    }

    @Override
    public void init(ClientWindow parent)
    {
        name = new JTextField(20);
        surname = new JTextField(20);
        company = new JTextField(20);
    }

    /** Конструктор окна добавления данных клиента
     *
```

```
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public AddDialogClient(JFrame owner, ClientWindow parent,
String title)
    {
        super(owner,parent,title);
    }
}
```

AddDialogManager.java

```
package Factory.gui;

import Factory.util.DateLabelFormatter;

import org.jdatepicker.impl.*;
import java.util.Date;
import java.util.Properties;
import javax.swing.*;

/**
 * Класс окна добавления данных контракта
 */
public class AddDialogContract extends DialogContract {

    @Override
    public void progress(ContractWindow parent) {
        setVisible(false);
        String[] arr = {description.getText(),
                        price.getText(),
                        (String) clients.getSelectedItem(),
                        (String) managers.getSelectedItem()};
        parent.addR(arr,(Date) dataBegin.getModel().getValue(),
(Date) dataEnd.getModel().getValue());

        JOptionPane.showMessageDialog(null, "Вы добавили новый
договор");
    }

    @Override
    public void init(ContractWindow parent)
    {
        description = new JTextField(20);
        price = new JTextField(20);
        clients = new JComboBox(parent.getClients());
        managers = new JComboBox(parent.getManagers());

        UtilDateModel model1 = new UtilDateModel();
        UtilDateModel model2 = new UtilDateModel();
```

```
        Properties p = new Properties();
        p.put("text.today", "Today");
        p.put("text.month", "Month");
        p.put("text.year", "Year");
        JDatePanelImpl datePanel1 = new JDatePanelImpl(model1, p);
        JDatePanelImpl datePanel2 = new JDatePanelImpl(model2, p);
        dataBegin = new JDatePickerImpl(datePanel1, new
DateLabelFormatter());
        dataEnd = new JDatePickerImpl(datePanel2, new
DateLabelFormatter());
    }

    /** Конструктор окна добавления данных контракта
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public AddDialogContract(JFrame owner, ContractWindow parent,
String title)
    {
        super(owner,parent,title);
    }
}
```

AddDialogWorkerContract.java

```
package Factory.gui;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Класс Диалогового окна добавления контрактов рабочему
 */
public class AddDialogWorkerContract extends JDialog
{
    /** Текстовое поле названия */
    protected JTextField name;

    /** Переменная корректности ввода */
    protected boolean check;

    /** Кнопка принять */
    private JButton ok;

    /** Кнопка отменить */
```

```
    private JButton cancel;

    /**
     * Выполнение манипуляций с данными
     *
     * @param parent - Объект класса приложения
     */
    public void progress(WorkerContractWindow parent)
    {
        setVisible(false);
        try
        {
            parent.addR(Integer.parseInt(name.getText()));
            JOptionPane.showMessageDialog(null, "Вы договор для
выполения рабонику");
        }
        catch (Exception ex)
        {
            JOptionPane.showMessageDialog(null, "Ошибка" +
ex.toString());

        }
    };

    /**
     * Инициализация
     *
     * @param parent - Объект класса приложения
     */
    public void init(WorkerContractWindow parent)
    {
        ok = new JButton("Принять");
        cancel = new JButton("Закрыть");
        name = new JTextField(20);
    };

    /**
     * Основной конструктор
     *
     * @param owner - JFrame приложения
     * @param parent - Объект класса приложения
     * @param title - Title окна
     */
    public AddDialogWorkerContract(JFrame owner,
WorkerContractWindow parent, String title)
    {
        super(owner, title, true);
        // Инит кнопок
        init(parent);
        check = false;
        ok.setEnabled(false);
```

```
        // Создание валидатора полей
        name.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                checker(name);
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                checker(name);
            }

            @Override
            public void changedUpdate(DocumentEvent e) {

            }
        });

        ok.addActionListener((e) -> progress(parent));
        cancel.addActionListener((e) -> dispose());
        JPanel mainp = new JPanel();

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2, 2, 2));

        panel.setSize(300, 100);

        // adds to the GridLayout
        panel.add(new Label("ID контракта"));
        panel.add(name);

        mainp.add(panel);
        add(BorderLayout.CENTER, mainp);
        JPanel but = new JPanel();
        but.add(ok);
        but.add(cancel);
        add(BorderLayout.SOUTH, but);
        setLocation(500, 250);
        setSize(500, 155);
        this.getRootPane().setDefaultButton(ok);

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ENTER"), "none");

this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("released ENTER"),
"press");
```

```
    }

    /**
     * Проверка поля на корректность введенных данных
     * @param field - проверяемое поле
     */
    protected void checker(JTextField field)
    {
        Pattern r = Pattern.compile("^[+-]?(([1-9][0-9]*)|(0))$");
        Matcher m = r.matcher(field.getText());
        if (m.matches()) {

field.setBorder(BorderFactory.createLineBorder(Color.GREEN));
            check = true;
        }
        else {

field.setBorder(BorderFactory.createLineBorder(Color.RED));
            check = false;
        }

        ok.setEnabled(check);
    }
}
```

EditDialogSpecialisation.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна изменения данных профессии
 */
public class EditDialogSpecialisation extends DialogSpecialisation
{
    @Override
    public void progress(SpecialisationWindow parent)
    {
        setVisible(false);
        int row = parent.dataSpecialisations.getSelectedRow();
        String[] arr = {parent.dataSpecialisations.getValueAt(row,
0).toString(), name.getText()};
        parent.dataSpecialisations.setValueAt(name.getText(), row,
1);
        parent.editR(arr);
    }

    @Override
    public void init(SpecialisationWindow parent)
    {
        int row = parent.dataSpecialisations.getSelectedRow();
        name = new
```

```
JTextField(parent.dataSpecialisations.getValueAt(row,
1).toString(), 20);
        checker(name);
    }

    /** Конструктор окна изменения данных профессии
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public EditDialogSpecialisation(JFrame owner,
SpecialisationWindow parent, String title)
    {
        super(owner, parent, title);
    };
}
```

EditDialogWorker.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна изменения данных рабочего
 */
public class EditDialogWorker extends DialogWorker
{
    @Override
    public void progress(WorkerWindow parent)
    {
        setVisible(false);
        int row = parent.dataWorkers.getSelectedRow();
        String[] arr = {parent.dataWorkers.getValueAt(row,
0).toString(), name.getText(), surname.getText(), exp.getText(),
(String) specs.getSelectedItem()};
        parent.dataWorkers.setValueAt(name.getText(), row, 1);
        parent.dataWorkers.setValueAt(surname.getText(), row, 2);
        parent.dataWorkers.setValueAt(exp.getText(), row, 3);
        parent.dataWorkers.setValueAt(specs.getSelectedItem(),
row, 4);
        parent.editR(arr);
    }

    @Override
    public void init(WorkerWindow parent)
    {
        int row = parent.dataWorkers.getSelectedRow();
        name = new JTextField(parent.dataWorkers.getValueAt(row,
1).toString(), 20);
        surname = new
JTextField(parent.dataWorkers.getValueAt(row, 2).toString(), 20);
```

```
        exp = new JTextField(parent.dataWorkers.getValueAt(row,
3).toString(), 20);
        specs = new JComboBox(parent.getSpecs());
        checker(0,name);
        checker(1,surname);
        checkerInt(2, exp);
    }

    /** Конструктор окна изменения данных рабочего
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public EditDialogWorker(JFrame owner, WorkerWindow parent,
String title)
    {
        super(owner, parent, title);
    };
}
```

EditDialogManager.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна изменения данных менеджера
 */
public class EditDialogManager extends DialogManager
{
    @Override
    public void progress(ManagerWindow parent)
    {
        setVisible(false);
        int row = parent.dataManagers.getSelectedRow();
        String[] arr = {parent.dataManagers.getValueAt(row,
0).toString(), name.getText(), surname.getText()};
        parent.dataManagers.setValueAt(name.getText(), row, 1);
        parent.dataManagers.setValueAt(surname.getText(), row, 2);
        parent.editR(arr);
    }

    @Override
    public void init(ManagerWindow parent)
    {
        int row = parent.dataManagers.getSelectedRow();
        name = new JTextField(parent.dataManagers.getValueAt(row,
1).toString(), 20);
        surname = new
JTextField(parent.dataManagers.getValueAt(row, 2).toString(), 20);
        checker(0,name);
```

```
        checker(1,surname);
    }

    /** Конструктор окна изменения данных менеджера
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public EditDialogManager(JFrame owner, ManagerWindow parent,
String title)
    {
        super(owner, parent, title);
    };
}
```

EditDialogClient.java

```
package Factory.gui;

import javax.swing.*;

/**
 * Класс окна изменения данных клиентов
 */
public class EditDialogClient extends DialogClient
{
    @Override
    public void progress(ClientWindow parent)
    {
        setVisible(false);
        int row = parent.dataClients.getSelectedRow();
        String[] arr = {parent.dataClients.getValueAt(row,
0).toString(), name.getText(), surname.getText(),
company.getText()};
        parent.dataClients.setValueAt(name.getText(), row, 1);
        parent.dataClients.setValueAt(surname.getText(), row, 2);
        parent.dataClients.setValueAt(company.getText(), row, 3);
        parent.editR(arr);
    }

    @Override
    public void init(ClientWindow parent)
    {
        int row = parent.dataClients.getSelectedRow();
        name = new JTextField(parent.dataClients.getValueAt(row,
1).toString(), 20);
        surname = new
JTextField(parent.dataClients.getValueAt(row, 2).toString(), 20);
        company = new
JTextField(parent.dataClients.getValueAt(row, 3).toString(), 20);
        checker(0,name);
        checker(1,surname);
```

```
        checker(2,company);
    }

    /** Конструктор окна изменения данных клиента
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public EditDialogClient(JFrame owner, ClientWindow parent,
String title)
    {
        super(owner, parent, title);
    };
}
```

EditDialogContract.java

```
package Factory.gui;

import Factory.util.DateLabelFormatter;
import org.jdatepicker.impl.JDatePanelImpl;
import org.jdatepicker.impl.JDatePickerImpl;
import org.jdatepicker.impl.UtilDateModel;

import javax.swing.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Properties;

/**
 * Класс окна изменения данных контракта
 */
public class EditDialogContract extends DialogContract
{
    @Override
    public void progress(ContractWindow parent)
    {
        setVisible(false);
        int row = parent.dataContracts.getSelectedRow();
        String[] arr = {parent.dataContracts.getValueAt(row,
0).toString(),
                description.getText(),
                price.getText(),
                (String) clients.getSelectedItem(),
                (String) managers.getSelectedItem()
        };
        parent.dataContracts.setValueAt(description.getText(),
row, 1);
        parent.dataContracts.setValueAt(price.getText(), row, 2);
        parent.dataContracts.setValueAt(clients.getSelectedItem(),
```

```
row, 3);

parent.dataContracts.setValueAt(managers.getSelectedItem(), row,
4);
        parent.dataContracts.setValueAt(new
SimpleDateFormat("yyyy-MM-dd").format((Date)
dataBegin.getModel().getValue()), row, 5);
        parent.dataContracts.setValueAt(new
SimpleDateFormat("yyyy-MM-dd").format((Date)
dataEnd.getModel().getValue()), row, 6);

        parent.editR(arr,(Date) dataBegin.getModel().getValue(),
(Date) dataEnd.getModel().getValue());
    }

    @Override
    public void init(ContractWindow parent)
    {
        int row = parent.dataContracts.getSelectedRow();
        description = new
JTextField(parent.dataContracts.getValueAt(row, 1).toString(),
20);
        price = new
JTextField(parent.dataContracts.getValueAt(row, 2).toString(),
20);
        clients = new JComboBox(parent.getClients());
        managers = new JComboBox(parent.getManagers());
        String dateValue = parent.dataContracts.getValueAt(row,
5).toString();
        Date date1 = null;
        Date date2 = null;
        try {
            date1 = new SimpleDateFormat("yyyy-mm-
dd").parse(dateValue);
            dateValue = parent.dataContracts.getValueAt(row,
6).toString();
            date2 = new SimpleDateFormat("yyyy-mm-
dd").parse(dateValue);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        UtilDateModel model1 = new UtilDateModel(date1);
        UtilDateModel model2 = new UtilDateModel(date2);
        Properties p = new Properties();
        p.put("text.today", "Today");
        p.put("text.month", "Month");
        p.put("text.year", "Year");
        JDatePanelImpl datePanel1 = new JDatePanelImpl(model1, p);
        JDatePanelImpl datePanel2 = new JDatePanelImpl(model2, p);
        dataBegin = new JDatePickerImpl(datePanel1, new
DateLabelFormatter());
```

```
        dataEnd = new JDatePickerImpl(datePanel2, new
DateLabelFormatter());
        checkerFloat(price);
    }

    /** Конструктор окна изменения данных контракта
     *
     * @param owner - JFrame приложения
     * @param parent - объект класса приложения
     * @param title - название окна
     */
    public EditDialogContract(JFrame owner, ContractWindow parent,
String title)
    {
        super(owner, parent, title);
    };
}
```

Package Factory.model

Client.java

```
package Factory.model;

import java.util.*;
import javax.persistence.*;

/**
 * Класс клиента завода по производству металлических изделий.
 * Наследник класса {@link Person}
 * @author Яловега Никита 9308
 * @version 0.1
 */
@Entity(name = "clients")
@Table(name = "clients")
public class Client extends Person
{
    /** Название компании клиента */
    @Column(name="company")
    private String company;

    /** Контракты клиента */
    @OneToMany(mappedBy = "client")
    private List<Contract> client_contracts;

    public Client() { }

    /**
     * Конструктор - создание нового объекта {@link Client}
     * @param name - имя
```

```
     * @param lastName - фамилия
     * @param company - компания
     */
    public Client(String name, String lastName, String company)
    {
        super(name, lastName);
        this.company = company;
    }

    /**
     * Функция получения значения поля {@link Client#company}
     * @return возвращает название компании клиента
     */
    public String getCompany()
    {
        return company;
    }

    /**
     * Процедура определения значения поля {@link Client#company}
     * @param newCompany - новая название компании клиенита
     */
    public void setCompany(String newCompany)
    {
        company = newCompany;
    }

    /**
     * Процедура добавления новых контрактов клиента
     * @param newContract - новый контракт
     */
    public void addContract(Contract newContract)
    {
        client_contracts.add(newContract);
        newContract.setClient(this); // связываем контракт с этим
клиентом
    }

    /**
     * Функция получения значения поля {@link
Client#client_contracts}
     * @return возвращает контракты клиента
     */
    public List<Contract> getContracts()
    {
        return client_contracts;
    }

    /**
     * Процедура удаления контрактов
     * @param c - контракт
```

```
     */
    public void removeContract(Contract c)
    {
        client_contracts.remove(c);
        c.setClient(null);
    }

    /**
     * Функция получения всей информации об объекте
     * @return  - массив строк с данными
     */
    public String[] toTableFormat()
    {
        return new String[] {String.valueOf(id), name, surname,
company};
    }
}
```

Contract.java

```
package Factory.model;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import javax.persistence.*;

/**
 * Класс контракта завода по производству металлических изделий.
 * @author Яловега Никита 9308
 * @version 0.1
*/
@Entity(name = "contracts")
@Table(name="contracts")
public class Contract
{
    /** Уникальный идентификатор контракта */
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    /** Описание условий контракта */
    @Column(name="description")
    private String description;

    /** Цена контракта */
    @Column(name="price")
    private double price;
```

```java
    /** Клиент, подписавший контракт */
    @ManyToOne
    @JoinColumn (name = "client_id")
    private Client client;

    /** Менеджер, подписавший контракт */
    @ManyToOne
    @JoinColumn (name = "manager_id")
    private Manager manager;

    /** Рабочие, выполняющие условия контракта */
    @ManyToMany(mappedBy = "contracts", fetch = FetchType.EAGER)
    private List<Employee> workers;

    /** Дата начала действия контракта */
    @Column(name="begin")
    private Date dateBegin;

    /** Дата окончания действия контракта */
    @Column(name="end")
    private Date dateEnd;

    /** Завершили ли выполнение договора */
    @Column(name="is_end")
    private boolean isEnd;

    /** Стандатный конструктор контракта {@link Contract} */
    public Contract() {}

    /**
     * Конструктор - создание нового объекта {@link Contract}
     */
    public Contract(String d, double p, Client c, Manager m,
List<Employee> w, Date b, Date e, boolean i)
    {
        this.description = d;
        this.price = p;
        this.client = c;
        this.manager = m;
        this.workers = w;
        this.dateBegin = b;
        this.dateEnd = e;
        this.isEnd = i;
    }

    /**
     * Метод получения значения поля {@link Contract#id}
     * @return возвращает уникальный идентификатор контракта
     */
    public int getId()
```

```
    {
        return id;
    }

    /**
     * Функция определения значения поля {@link Contract#id}
     * @param newID - новый идентификатор
     */
    public void setId(int newID)
    {
        id = newID;
    }

    /**
     * Метод определения значения поля {@link
Contract#description}
     * @param newDescription - описание контракта
     */
    public void setDescription(String newDescription)
    {
        description = newDescription;
    }

    /**
     * Метод получения значения поля {@link Contract#description}
     * @return возвращает описание контракта
     */
    public String getDescription()
    {
        return description;
    }

    /**
     * Метод определения значения поля {@link Contract#price}
     * @param newPrice - цена контракта
     */
    public void setPrice(double newPrice)
    {
        price = newPrice;
    }

    /**
     * Метод получения значения поля {@link Contract#price}
     * @return возвращает цену контракта
     */
    public double getPrice()
    {
        return price;
    }

    /**
```

```java
 * Метод определения значения поля {@link Contract#client}
 * @param newClient - новый клиент контракта
 */
public void setClient(Client newClient)
{
    client = newClient;
}

/**
 * Метод получения значения поля {@link Contract#client}
 * @return возвращает клиента
 */
public Client getClient()
{
    return client;
}

/**
 * Метод определения значения поля {@link Contract#manager}
 * @param newManager - новый менеджен контракта
 */
public void setManager(Manager newManager)
{
    manager = newManager;
}

/**
 * Метод получения значения поля {@link Contract#manager}
 * @return возвращает менеджера
 */
public Manager getManager()
{
    return manager;
}

/**
 * Метод определения значения поля {@link Contract#dateBegin}
 * @param newDate - новая дата
 */
public void setDateBegin(Date newDate)
{
    dateBegin = newDate;
}

/**
 * Метод получения значения поля {@link Contract#dateBegin}
 * @return возвращает дату начала
 */
public Date getDateBegin()
{
    return dateBegin;
```

```
    }

    /**
     * Метод определения значения поля {@link Contract#dateEnd}
     * @param newDate - новая дата
     */
    public void setDateEnd(Date newDate)
    {
        dateEnd = newDate;
    }

    /**
     * Метод получения значения поля {@link Contract#dateEnd}
     * @return возвращает дату начала
     */
    public Date getDateEnd()
    {
        return dateEnd;
    }

    /**
     * Метод добавления рабочего, который выполняет контракт
{@link Contract#workers}
     * @param newWorker - новый рабочий
     */
    public void addWorker(Employee newWorker)
    {
        workers.add(newWorker);
    }

    /**
     * Метод удаления рабочего, который выполняет контракт {@link
Contract#workers}
     * @param worker - рабочий
     */
    public void removeWorker(Employee worker)
    {
        workers.remove(worker);
    }

    /**
     * Метод установки рабочих, которые выполнят контракт {@link
Contract#workers}
     * @param w - список рабочих
     */
    public void setWorkers(List<Employee> w)
    {
        workers = w;
    }

    /**
```

```
     * Метод получения значения поля {@link Contract#workers}
     * @return возвращает всех рабочих, выполняющих контракт
     */
    public List<Employee> getWorkers()
    {
        return workers;
    }

    /**
     * Метод получения значения поля {@link Contract#isEnd}
     * @return возвращает состояние договора
     */
    public boolean getIsEnd()
    {
        return isEnd;
    }

    /**
     * Функция определения значения поля {@link Contract#id}
     * @param i - новое состояние
     */
    public void setIsEnd(boolean i)
    {
        isEnd = i;
    }

    /**
     * Функция получения всей информации об объекте
     * @return  - массив строк с данными
     */
    public String[] toTableFormat()
    {
        DateFormat df = new SimpleDateFormat("yyyy-MM-dd");

        return new String[] {
                String.valueOf(id),
                description,
                String.valueOf(price),
                String.valueOf(client.getId()) + " " +
client.getName() + " " + client.getSurname(),
                String.valueOf(manager.getId()) + " " +
manager.getName() + " " + manager.getSurname(),
                df.format(dateBegin),
                df.format(dateEnd),
                isEnd ? "Выполнено" : "В процессе"
        };
    }

}
```

Employee.java

```java
package Factory.model;

import java.util.List;
import javax.persistence.*;

/**
 * Класс сотрудника завода по производству металлических изделий.
 * Наследник класса {@link Person}
 * @author Яловега Никита 9308
 * @version 0.1
 */
@Entity(name = "employees")
@Table(name = "employees")
public class Employee extends Person
{
    /** Поле опыта работы */
    @Column(name="exp")
    private int exp;

    /** Поле профессии рабочего */
    @ManyToOne
    @JoinColumn (name="specialisation_id")
    private Specialisation specialisation;

    /** Контракты, в которых участвует рабочий */
    @ManyToMany(fetch = FetchType.EAGER, cascade =
CascadeType.ALL)
    @JoinTable(
            name = "employees_contracts",
            joinColumns = { @JoinColumn(name = "workers_id",
referencedColumnName = "id" )},
            inverseJoinColumns = { @JoinColumn(name =
"contracts_id", referencedColumnName = "id") }
    )
    private List<Contract> contracts;

    /**
     * Стандартный контруктор
     */
    public Employee(){}

    /**
     * Конструктор - создание нового объекта Employee
     * @param name - имя
     * @param lastName - фамилия
     * @param exp - опыт работы
     * @param specialisation - профессия
     */
    public Employee(String name, String lastName, int exp,
```

```
Specialisation specialisation)
    {
        super(name, lastName);
        this.exp = exp;
        this.specialisation = specialisation;
        this.contracts = null;
    }

    /**
     * Функция получения значения поля {@link Employee#exp}
     * @return возвращает опыт работы рабочего
     */
    public int getExp()
    {
        return exp;
    }

    /**
     * Процедура определения значения поля {@link Employee#exp}
     * @param newexp - новая фамилия человека
     */
    public void setExp(int newexp)
    {
        exp = newexp;
    }

    /**
     * Процедура определения значения поля {@link
Employee#specialisation}
     * @param d - профессия сотрудника
     */
    public void setSpecialisation(Specialisation d)
    {
        specialisation = d;
    }

    /**
     * Функция получения значения поля {@link
Employee#specialisation}
     * @return возвращает профессию сотрудника
     */
    public Specialisation getSpecialisation()
    {
        return specialisation;
    }

    /**
     * Процедура добавления новых контрактов рабочему.
     * @param newContract - новый контракт, который выполняет
рабочий
     */
```

```
    public void addContract(Contract newContract)
    {
        contracts.add(newContract);
        newContract.addWorker(this); // добавляем в контракт
рабочего
    }

    /**
     * Функция получения значения поля {@link Employee#contracts}
     * @return возвращает контракты, который выполняет рабочий
     */
    public List<Contract> getContracts()
    {
        return contracts;
    }

    /**
     * Процедура удаления контрактов, которые выполняет рабочий
     * @param c - контракт
     */
    public void removeContract(Contract c)
    {
        contracts.remove(c);
        c.removeWorker(this);
    }

    /**
     * Функция получения всей информации об объекте
     * @return  - массив строк с данными
     */
    public String[] toTableFormat()
    {
        return new String[] {String.valueOf(id), name, surname,
String.valueOf(exp), specialisation.getName()};
    }

}
```

Manager.java

```
package Factory.model;

import java.util.List;
import javax.persistence.*;

/**
 * Класс менеджера завода по производству металлических изделий.
 * Наследник класса {@link Person}
 * @author Яловега Никита 9308
 * @version 0.1
 */
```

```java
@Entity(name = "managers")
@Table(name = "managers")
public class Manager extends Person
{
    /** Контракты, в которые подписывал менеджер */
    @OneToMany(mappedBy = "manager")
    private List<Contract> contracts;

    public Manager() {}

    /**
     * Конструктор - создание нового объекта Manager
     * @param name - имя
     * @param lastName - фамилия
     */
    public Manager(String name, String lastName)
    {
        super(name, lastName);
    }

    /**
     * Процедура добавления новых контрактов рабочему.
     * @param newContract - новый контракт, который выполняет
рабочий
     */
    public void addContract(Contract newContract)
    {
        contracts.add(newContract);
        newContract.setManager(this); // добавляем в контракт
рабочего
    }

    /**
     * Функция получения значения поля {@link Manager#contracts}
     * @return возвращает контракты, который выполняет рабочий
     */
    public List<Contract> getContracts()
    {
        return contracts;
    }

    /**
     * Процедура удаления контрактов, которые выполняет рабочий
     * @param c - контракт
     */
    public void removeContract(Contract c)
    {
        contracts.remove(c);
        c.setManager(null);
    }
```

```
    /**
     * Процедура установки контрактов, которые выполняет рабочий
     * @param c - список контракт
     */
    public void setContract(List<Contract> c)
    {
        contracts = c;
    }

    /**
     * Функция получения всей информации об объекте
     * @return  - массив строк с данными
     */
    public String[] toTableFormat()
    {
        return new String[] {String.valueOf(id), name, surname};
    }
}
```

Person.java

```
package Factory.model;

import javax.persistence.*;

/**
 * Класс человека
 * @author Яловега Никита 9308
 * @version 0.1
 */
@MappedSuperclass
public class Person
{
    /** Поле уникального идентификатора */
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    protected int id;

    /** Поле имени человека */
    @Column(name="name")
    protected String name;

    /** Поле фимилии человека */
    @Column(name="surname")
    protected String surname;

    /** Стандартный контруктор человека */
    public Person() {}

    /**
```

```
 * Конструктор - создание нового объекта Person
 * @param name - имя
 * @param surname - фамилия
 */
public Person(String name, String surname)
{
    this.name = name;
    this.surname = surname;
}

/**
 * Функция получения значения поля {@link Person#id}
 * @return возвращает уникальный идентификатор человека
 */
public int getId()
{
    return id;
}

/**
 * Функция определения значения поля {@link Person#id}
 * @param newID - новый идентификатор пользователя
 */
public void setId(int newID)
{
    id = newID;
}

/**
 * Функция получения значения поля {@link Person#name}
 * @return возвращает имя человека
 */
public String getName()
{
    return name;
}

/**
 * Процедура определения значения поля {@link Person#name}
 * @param newName - новое имя человека
 */
public void setName(String newName)
{
    name = newName;
}

/**
 * Функция получения значения поля {@link Person#surname}
 * @return возвращает фамилию человека
 */
public String getSurname()
```

```
    {
        return surname;
    }

    /**
     * Процедура определения значения поля {@link Person#name}
     * @param newSurname - новая фамилия человека
     */
    public void setSurname(String newSurname)
    {
        surname = newSurname;
    }
}
```

Specialisation.java

```
package Factory.model;

import javax.persistence.*;
import java.util.List;

/**
 * Класс специализации работника
 * @author Яловега Никита 9308
 * @version 0.1
 */
@Entity(name = "specialisations")
@Table(name = "specialisations")
public class Specialisation
{

    /** Уникальный идентификатор профессии */
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    /** Название профессии */
    @Column(name="name")
    private String name;

    /** Рабочие данной профессии */
    @OneToMany(mappedBy = "specialisation", cascade =
CascadeType.ALL, orphanRemoval = true)
    private List<Employee> employees;

    public Specialisation(){}

    public Specialisation(String name)
    {
        this.name = name;
```

```
    }

    /**
     * Метод получения значения поля {@link Specialisation#id}
     * @return возвращает уникальный идентификатор
     */
    public int getID()
    {
        return id;
    }

    /**
     * Метод определения значения поля {@link Specialisation#id}
     * @param newID - новый идентификатор
     */
    public void setID(int newID)
    {
        id = newID;
    }

    /**
     * Метод получения значения поля {@link Specialisation#name}
     * @return возвращает название специализации
     */
    public String getName()
    {
        return name;
    }

    /**
     * Метод определения значения поля {@link Specialisation#name}
     * @param newName - новое название специализации
     */
    public void setName(String newName)
    {
        name = newName;
    }

    /**
     * Метод добавления новых работников {@link
Specialisation#employees}
     * @param newEmployee - новый работник
     */
    public void addEmployee(Employee newEmployee)
    {
        employees.add(newEmployee);
        // связываем сотрудника с этим отделом
        newEmployee.setSpecialisation(this);
    }

    /**
```

```
     * Метод получения значения поля {@link
Specialisation#employees}
     * @return Работники данной профессии
     */
    public List<Employee> getEmployees()
    {
        return employees;
    }

    /**
     * Метод удаления работников из профессии {@link
Specialisation#employees}
     * @param e - работник, которого нужно убрать
     */
    public void removeEmployee(Employee e)
    {
        employees.remove(e);
    }

    /**
     * Функция получения всей информации об объекте
     * @return  - массив строк с данными
     */
    public String[] toTableFormat()
    {
        return new String[] {String.valueOf(getID()), getName()};
    }
}
```

Package Factory.service

SpecialisationService.java

```
package Factory.service;

import Factory.dao.SpecialisationDAO;
import Factory.model.*;
import java.util.List;

public class SpecialisationService
{
    private static SpecialisationDAO specialisationDao;

    public SpecialisationService()
    {
        specialisationDao = new SpecialisationDAO();
    }

    public void persist(Specialisation entity) {
        specialisationDao.openCurrentSessionwithTransaction();
        specialisationDao.persist(entity);
```

```
        specialisationDao.closeCurrentSessionwithTransaction();
    }

    public void update(Specialisation entity) {
        specialisationDao.openCurrentSessionwithTransaction();
        specialisationDao.update(entity);
        specialisationDao.closeCurrentSessionwithTransaction();
    }

    public Specialisation findById(int id)
    {
        specialisationDao.openCurrentSession();
        Specialisation manager = specialisationDao.findById(id);
        specialisationDao.closeCurrentSession();
        return manager;
    }

    public Specialisation findByName(String name)
    {
        specialisationDao.openCurrentSession();
        Specialisation manager =
specialisationDao.findByName(name);
        specialisationDao.closeCurrentSession();
        return manager;
    }

    public void delete(int id) {
        specialisationDao.openCurrentSessionwithTransaction();
        Specialisation manager = specialisationDao.findById(id);
        specialisationDao.delete(manager);
        specialisationDao.closeCurrentSessionwithTransaction();
    }

    public List<Specialisation> findAll() {
        specialisationDao.openCurrentSession();
        List<Specialisation> managers =
specialisationDao.findAll();
        specialisationDao.closeCurrentSession();
        return managers;
    }

    public void deleteAll() {
        specialisationDao.openCurrentSessionwithTransaction();
        specialisationDao.deleteAll();
        specialisationDao.closeCurrentSessionwithTransaction();
    }

    public SpecialisationDAO specialisationDao()
    {
        return specialisationDao;
```

```
    }
}
```

ManagerService.java

```
package Factory.service;

import Factory.dao.ManagerDAO;
import Factory.model.Manager;
import java.util.List;

public class ManagerService
{
    private static ManagerDAO managerDao;

    public ManagerService()
    {
        managerDao = new ManagerDAO();
    }

    public void persist(Manager entity) {
        managerDao.openCurrentSessionwithTransaction();
        managerDao.persist(entity);
        managerDao.closeCurrentSessionwithTransaction();
    }

    public void update(Manager entity) {
        managerDao.openCurrentSessionwithTransaction();
        managerDao.update(entity);
        managerDao.closeCurrentSessionwithTransaction();
    }

    public Manager findById(int id) {
        managerDao.openCurrentSession();
        Manager manager = managerDao.findById(id);
        managerDao.closeCurrentSession();
        return manager;
    }

    public void delete(int id) {
        managerDao.openCurrentSessionwithTransaction();
        Manager manager = managerDao.findById(id);
        managerDao.delete(manager);
        managerDao.closeCurrentSessionwithTransaction();
    }

    public List<Manager> findAll() {
        managerDao.openCurrentSession();
        List<Manager> managers = managerDao.findAll();
        managerDao.closeCurrentSession();
        return managers;
    }
```

```
    public void deleteAll() {
        managerDao.openCurrentSessionwithTransaction();
        managerDao.deleteAll();
        managerDao.closeCurrentSessionwithTransaction();
    }

    public ManagerDAO managerDao()
    {
        return managerDao;
    }
}
```

EmployeeService.java

```
package Factory.service;

import Factory.dao.EmployeeDAO;
import Factory.model.*;
import java.util.List;

public class EmployeeService
{
    private static EmployeeDAO employeeDao;

    public EmployeeService()
    {
        employeeDao = new EmployeeDAO();
    }

    public void persist(Employee entity) {
        employeeDao.openCurrentSessionwithTransaction();
        employeeDao.persist(entity);
        employeeDao.closeCurrentSessionwithTransaction();
    }

    public void update(Employee entity) {
        employeeDao.openCurrentSessionwithTransaction();
        employeeDao.update(entity);
        employeeDao.closeCurrentSessionwithTransaction();
    }

    public Employee findById(int id) {
        employeeDao.openCurrentSession();
        Employee manager = employeeDao.findById(id);
        employeeDao.closeCurrentSession();
        return manager;
    }

    public void delete(int id) {
        employeeDao.openCurrentSessionwithTransaction();
        Employee manager = employeeDao.findById(id);
```

```
        employeeDao.delete(manager);
        employeeDao.closeCurrentSessionwithTransaction();
    }

    public List<Employee> findAll() {
        employeeDao.openCurrentSession();
        List<Employee> managers = employeeDao.findAll();
        employeeDao.closeCurrentSession();
        return managers;
    }

    public void deleteAll() {
        employeeDao.openCurrentSessionwithTransaction();
        employeeDao.deleteAll();
        employeeDao.closeCurrentSessionwithTransaction();
    }

    public EmployeeDAO employeeDao()
    {
        return employeeDao;
    }
}
```

ContractService.java

```
package Factory.service;

import Factory.dao.ContractDAO;
import Factory.model.*;

import java.util.Date;
import java.util.List;

public class ContractService
{
    private static ContractDAO contractDao;

    public ContractService()
    {
        contractDao = new ContractDAO();
    }

    public void persist(Contract entity) {
        contractDao.openCurrentSessionwithTransaction();
        contractDao.persist(entity);
        contractDao.closeCurrentSessionwithTransaction();
    }

    public void update(Contract entity) {
        contractDao.openCurrentSessionwithTransaction();
        contractDao.update(entity);
```

```
        contractDao.closeCurrentSessionwithTransaction();
    }

    public void setEnd(int id)
    {
        contractDao.openCurrentSessionwithTransaction();
        Contract contract = contractDao.findById(id);
        contract.setIsEnd(true);
        contractDao.update(contract);
        contractDao.closeCurrentSessionwithTransaction();
    }

    public Contract findById(int id) {
        contractDao.openCurrentSession();
        Contract contract = contractDao.findById(id);
        contractDao.closeCurrentSession();
        return contract;
    }

    public void delete(int id) {
        contractDao.openCurrentSessionwithTransaction();
        Contract contract = contractDao.findById(id);
        contractDao.delete(contract);
        contractDao.closeCurrentSessionwithTransaction();
    }

    public List<Contract> findAll()
    {
        contractDao.openCurrentSession();
        List<Contract> contracts = contractDao.findAll();
        contractDao.closeCurrentSession();
        return contracts;
    }

    public int findNew(Date date)
    {
        contractDao.openCurrentSession();
        int num = contractDao.findNew(date);
        contractDao.closeCurrentSession();
        return num;
    }

    public int findFinish(Date date)
    {
        contractDao.openCurrentSession();
        int num = contractDao.findFinish(date).size();
        contractDao.closeCurrentSession();
        return num;
    }
```

```
    public float findFinishIncome(Date date)
    {
        contractDao.openCurrentSession();
        float income = 0;
        for (Contract a : contractDao.findFinish(date))
            income += a.getPrice();
        contractDao.closeCurrentSession();
        return income;
    }

    public List<Contract> findOutdated() {
        contractDao.openCurrentSession();
        List<Contract> contracts = contractDao.findOutdated();
        contractDao.closeCurrentSession();
        return contracts;
    }

    public List<Contract> findByClientId(int id)
    {
        contractDao.openCurrentSession();
        List<Contract> contracts = contractDao.findByClientId(id);
        contractDao.closeCurrentSession();
        return contracts;
    }

    public List<Contract> findOutdated(int id)
    {
        contractDao.openCurrentSession();
        List<Contract> contracts = contractDao.findOutdated(id);
        contractDao.closeCurrentSession();
        return contracts;
    }

    public List<Contract> findTimePeriod(Date btime, Date etime)
    {
        contractDao.openCurrentSession();
        List<Contract> contracts =
contractDao.findTimePeriod(btime, etime);
        contractDao.closeCurrentSession();
        return contracts;
    }

    public List<Contract> findTimePeriod(Date btime, Date etime,
int id)
    {
        contractDao.openCurrentSession();
        List<Contract> contracts =
contractDao.findTimePeriod(btime, etime, id);
        contractDao.closeCurrentSession();
        return contracts;
    }
```

```
    public void deleteAll() {
        contractDao.openCurrentSessionwithTransaction();
        contractDao.deleteAll();
        contractDao.closeCurrentSessionwithTransaction();
    }

    public ContractDAO contractDao()
    {
        return contractDao;
    }
}
```

ClientService.java

```
package Factory.service;

import Factory.dao.ClientDAO;
import Factory.model.*;
import java.util.List;

public class ClientService
{
    private static ClientDAO clientDao;

    public ClientService()
    {
        clientDao = new ClientDAO();
    }

    public void persist(Client entity) {
        clientDao.openCurrentSessionwithTransaction();
        clientDao.persist(entity);
        clientDao.closeCurrentSessionwithTransaction();
    }

    public void update(Client entity) {
        clientDao.openCurrentSessionwithTransaction();
        clientDao.update(entity);
        clientDao.closeCurrentSessionwithTransaction();
    }

    public Client findById(int id) {
        clientDao.openCurrentSession();
        Client contract = clientDao.findById(id);
        clientDao.closeCurrentSession();
        return contract;
    }

    public void delete(int id) {
        clientDao.openCurrentSessionwithTransaction();
```

```
        Client contract = clientDao.findById(id);
        clientDao.delete(contract);
        clientDao.closeCurrentSessionwithTransaction();
    }

    public List<Client> findAll() {
        clientDao.openCurrentSession();
        List<Client> contracts = clientDao.findAll();
        clientDao.closeCurrentSession();
        return contracts;
    }

    public List<Client> findByManagerId(int id) {
        clientDao.openCurrentSession();
        List<Client> contracts = clientDao.findByManagerId(id);
        clientDao.closeCurrentSession();
        return contracts;
    }

    public void deleteAll() {
        clientDao.openCurrentSessionwithTransaction();
        clientDao.deleteAll();
        clientDao.closeCurrentSessionwithTransaction();
    }

    public ClientDAO clientDao()
    {
        return clientDao;
    }
}
```

Package Factory.util

ReportUtil.java

```
package Factory.util;

import net.sf.jasperreports.engine.*;
import net.sf.jasperreports.engine.data.JRXmlDataSource;
import net.sf.jasperreports.engine.design.JasperDesign;
import net.sf.jasperreports.engine.xml.JRXmlLoader;

import java.io.File;

/**
 * Класс для генерации отчетов
 */
public class ReportUtil
{
```

```
    /**
     * Метод генерации отчетов в формате PDF.
     * @param datasource Имя файла XML с данными
     * @param xpath Директория до полей с данными
     * @param template Имя файла шаблона .jrxml
     * @param resultpath Имя файла, в который будет помещен отчет
     */
    public static void print(String datasource, String xpath,
String template, String resultpath) throws JRException
    {
        File reportPattern = new File(template);

        JasperDesign jasperDesign =
JRXmlLoader.load(reportPattern);
        JasperReport jasperReport =
JasperCompileManager.compileReport(jasperDesign);
        JRDataSource jr = new JRXmlDataSource(datasource, xpath);
        JasperPrint jasperPrint =
JasperFillManager.fillReport(jasperReport, null, jr);
        JasperExportManager.exportReportToPdfFile(jasperPrint,
resultpath);
    }
}
```

HibernateSessionFactoryUtil.java

```
package Factory.util;

import Factory.model.*;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;

public class HibernateSessionFactoryUtil
{
    private static SessionFactory sessionFactory;

    private HibernateSessionFactoryUtil() {}

    public static SessionFactory getSessionFactory()
    {
        if (sessionFactory == null)
        {
            Configuration configuration = new Configuration()
                    .configure("hibernate.cfg.xml")
                    .addAnnotatedClass(Person.class)
                    .addAnnotatedClass(Employee.class)
                    .addAnnotatedClass(Specialisation.class)
                    .addAnnotatedClass(Manager.class)
                    .addAnnotatedClass(Client.class)
```

```
                    .addAnnotatedClass(Contract.class);
            StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder().applySettings(configuration.getPr
operties());
            sessionFactory =
configuration.buildSessionFactory(builder.build());
        }

        return sessionFactory;
    }
}
```

DateLabelFormatter.java

```
package Factory.util;

import java.util.*;
import java.text.*;
import javax.swing.JFormattedTextField.AbstractFormatter;


public class DateLabelFormatter extends AbstractFormatter {

    private final String datePattern = "yyyy-MM-dd";
    private final SimpleDateFormat dateFormatter = new
SimpleDateFormat(datePattern);

    @Override
    public Object stringToValue(String text) throws ParseException
{
        return dateFormatter.parseObject(text);
    }

    @Override
    public String valueToString(Object value){
        if (value != null) {
            Calendar cal = (Calendar) value;
            return dateFormatter.format(cal.getTime());
        }

        return "";
    }

}
```