



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

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**
МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/05 Современные интеллектуальные
программно-аппаратные комплексы.**

О Т Ч Е Т

по лабораторной работе № 2

Название: Создание прототипа ИС на базе MongoDB

Дисциплина: Распределённые базы данных

Студент

ИУ6-11М

(Группа)

(Подпись, дата)

И.С. Марчук

(И.О. Фамилия)

Преподаватель

М.М. Фомин

(Подпись, дата)

(И.О. Фамилия)

Москва, 2023

Цель работы: научиться создавать БД в MongoDB.

Вариант: 8

Предметная область для практических заданий: **Сотовая связь**



Рисунок 1 – Предметная область «Сотовая связь»

Задание: разработать фрагмент базы данных системы, предметную область которой кратко описана в варианте. Конкретную предметную область можно получить, взяв вариант, совпадающий с вашим порядковым номером в списке группы. Надо сказать, что приведенные схемы весьма общи, а иногда и не очень корректны, поэтому Вы можете изменить их по вашему усмотрению, оставаясь, конечно, в рамках предметной области. Для вашей предметной области необходимо **выделить минимум 2 бизнес-процесса**, которые поддаются автоматизации (это не значит, что вы должны их автоматизировать). По этим **бизнес-процессам** надо **выделить сущности предметной области**, которые создают костяк информационной модели предметной области. Далее нужно **прорисовать примерную схему, показывающую связи сущностей**. Не надо прорабатывать подробную схему таблиц как при работе с реляционной БД, да и можно ли говорить о «схеме» в NoSQL СУБД... Вам предлагается **создать интерфейсы к системе учета данных вашей предметной области с минимальным функционалом:**

- ввод и сохранение данных о базовых сущностях (можно просто скрипт создающий базу данных и заполняющую ее тестовыми значениями);

- поиск данных о базовых сущностях в соответствии с построенными бизнес-процессами (необходимо разработать интерфейс для поиска сущностей и скрипты выполняющие этот поиск);

- вывод и отображение данных хранящихся в БД.

Данные должны храниться в MongoDB, а инструментарий по построению интерфейса выбирается на ваше усмотрение.

Отчёт по лабораторной работе должен включать:

- схемы бизнес процессов;
- схему взаимодействия сущностей;
- тексты скриптов для создания и заполнения коллекций;
- тексты скриптов для поиска и выдачи данных по базовым сущностям;
- скриншоты интерфейсов и заполненных коллекций.

Ход работы:

У меня уже была установлена среда IntelliJIdea, поэтому я решил разработать прототип ИС в ней на языке java. Структура созданного проекта представлена на рисунке 2.

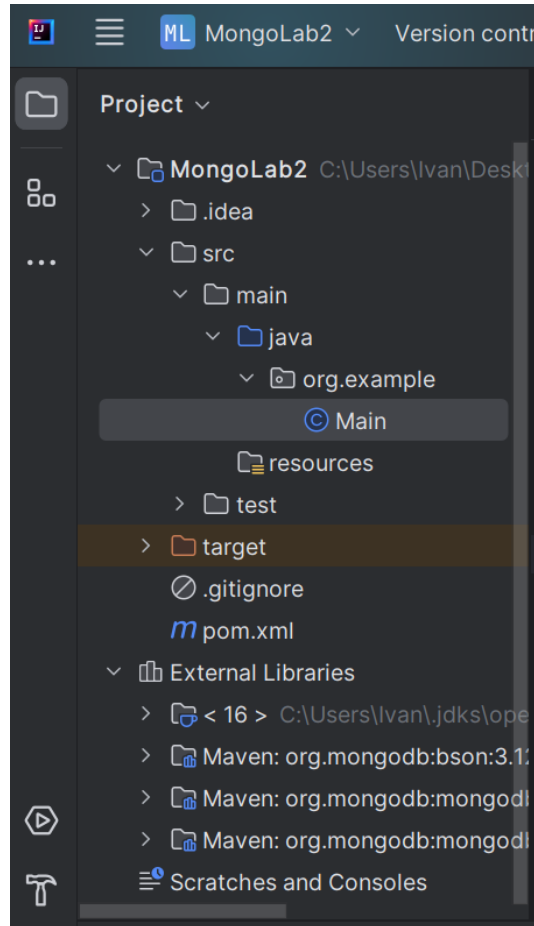
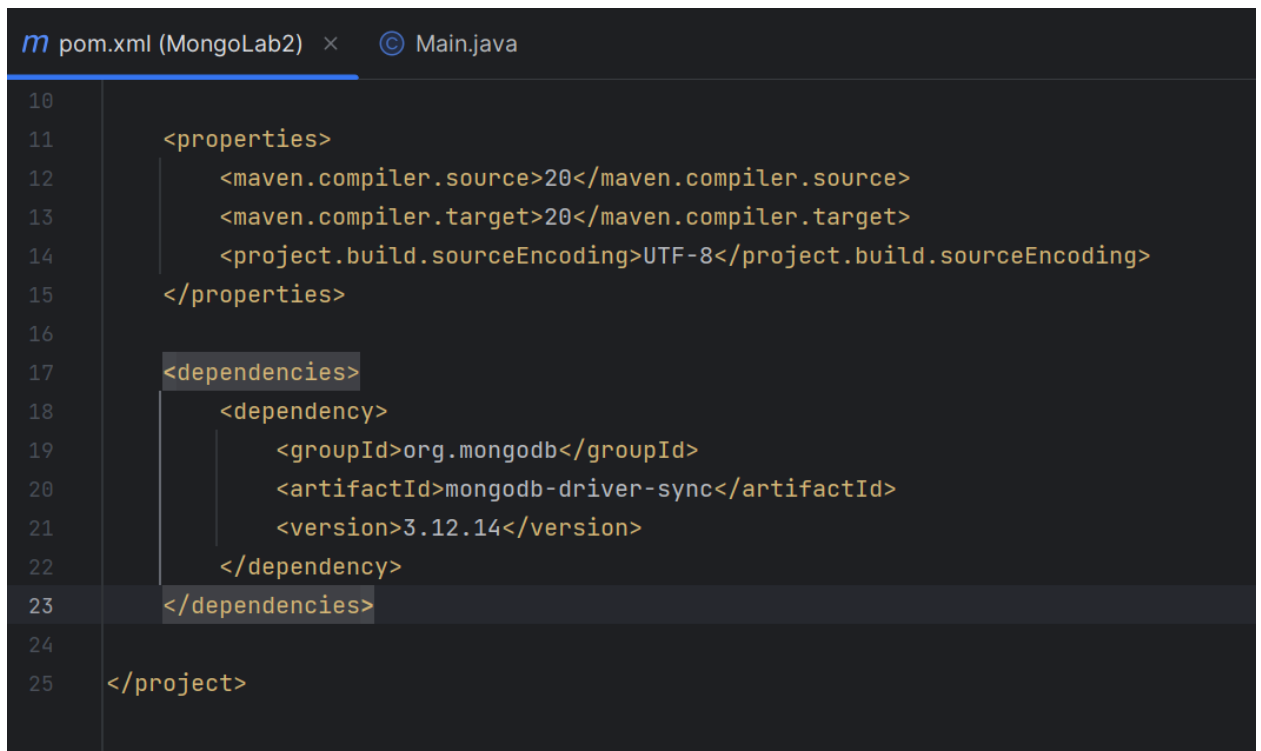


Рисунок 2 – Структура созданного проекта

Я создал проект со сборщиком maven и добавил в него библиотеку для работы с mongodb. Скриншот настройки Maven представлен на рисунке 3.



```
m pom.xml (MongoLab2) x Main.java
10
11 <properties>
12     <maven.compiler.source>20</maven.compiler.source>
13     <maven.compiler.target>20</maven.compiler.target>
14     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
15 </properties>
16
17 <dependencies>
18     <dependency>
19         <groupId>org.mongodb</groupId>
20         <artifactId>mongodb-driver-sync</artifactId>
21         <version>3.12.14</version>
22     </dependency>
23 </dependencies>
24
25 </project>
```

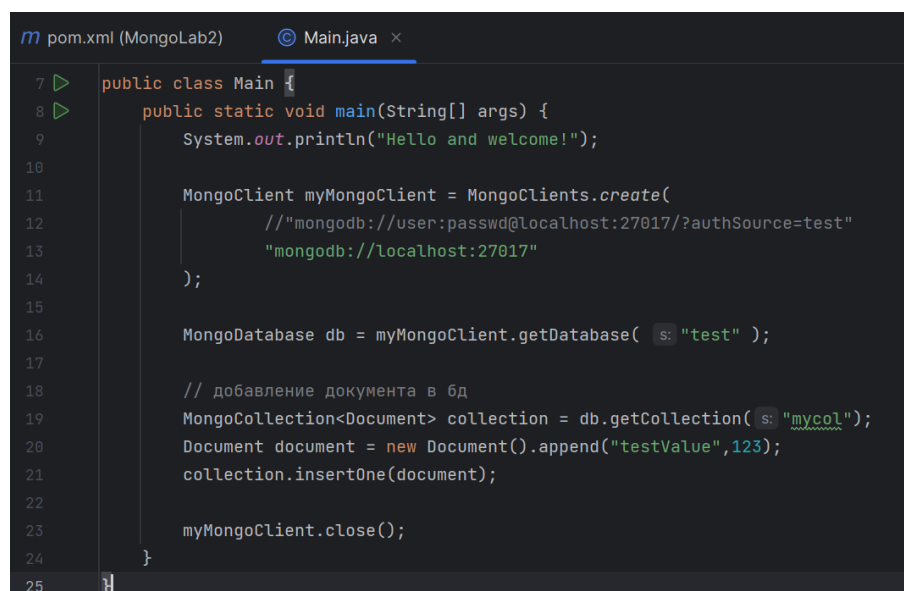
Рисунок 3 – Зависимость библиотеки MongoDB в файле Maven

Для дальнейшей работы необходимо было запустить на компьютере сервер с MongoDB. Я сделал это через консоль командой:

```
"c:\Program Files\MongoDB\Server\7.0\bin\mongod.exe" --
dbpath="c:\mongodata\db"
```

Доступ к серверу происходил по адресу «mongodb://localhost:27017»

Затем я написал код по добавлению тестовой коллекции с записью в БД (рисунок 4).



```
m pom.xml (MongoLab2) Main.java x
7 public class Main {
8     public static void main(String[] args) {
9         System.out.println("Hello and welcome!");
10
11         MongoClient myMongoClient = MongoClient.create(
12             //"mongodb://user:passwd@localhost:27017/?authSource=test"
13             "mongodb://localhost:27017"
14         );
15
16         MongoDBDatabase db = myMongoClient.getDatabase( s: "test" );
17
18         // добавление документа в бд
19         MongoCollection<Document> collection = db.getCollection( s: "mycol");
20         Document document = new Document().append("testValue",123);
21         collection.insertOne(document);
22
23         myMongoClient.close();
24     }
25 }
```

Рисунок 4 – Пример программы добавляющей документ в коллекцию

Для удобства пока я работал я активно пользовался программой MongoDB compass. В ней я и увидел созданные коллекции «test» и документ «mycol» (Рисунок 5).

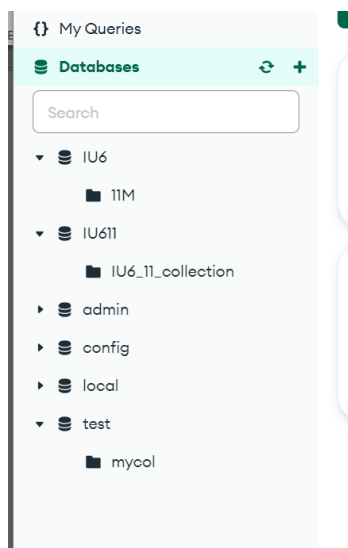


Рисунок 5 – Содержимое локальной базы данных

Дальнейшая работа по созданию ИС требовала проектирования. Я построил схемы бизнес-процессов, происходящих в предметной области по заданию. Схемы бизнес-процессов (IDEF0) представлены на рисунках 6-8.

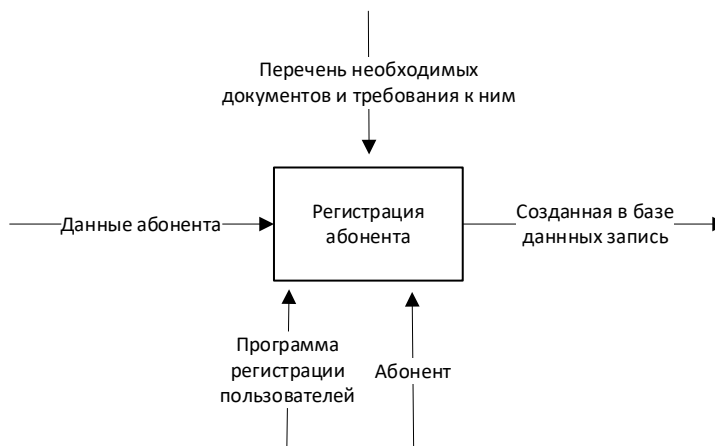


Рисунок 6 – Схема бизнес-процесса «Регистрация абонента»

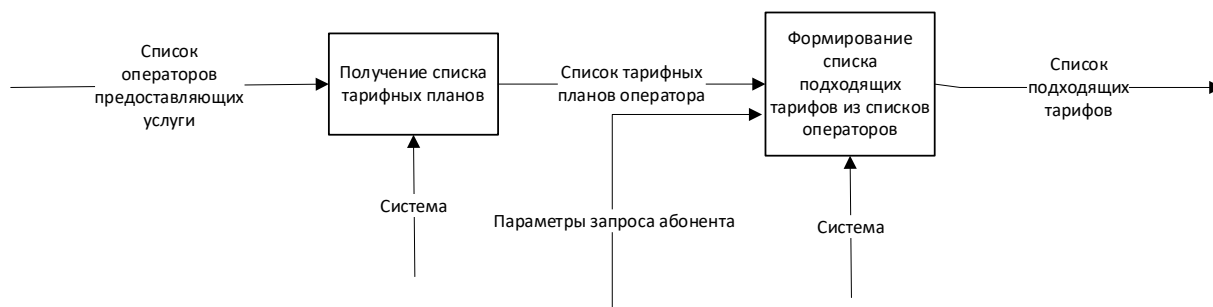


Рисунок 7 – Схема бизнес-процесса «Просмотр тарифных планов различных операторов»

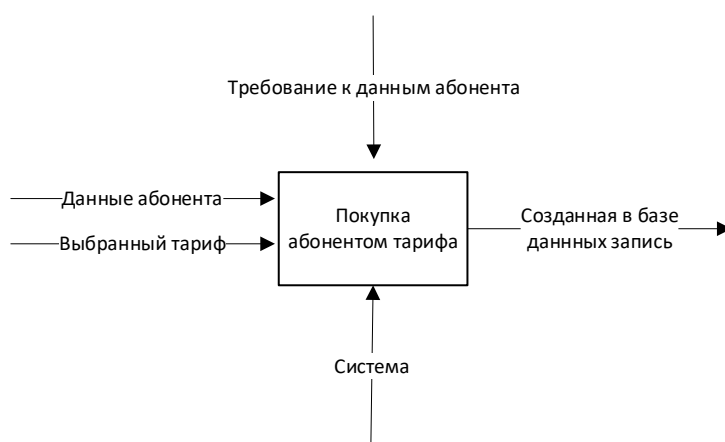


Рисунок 8 – Схема бизнес-процесса «Покупка абонентом тарифа» (требованиями могут быть, например отсутствие задолженности)

Для создания не реляционной базы данных я спроектировал схему взаимодействия сущностей (UML диаграмма классов) (рисунок 9):

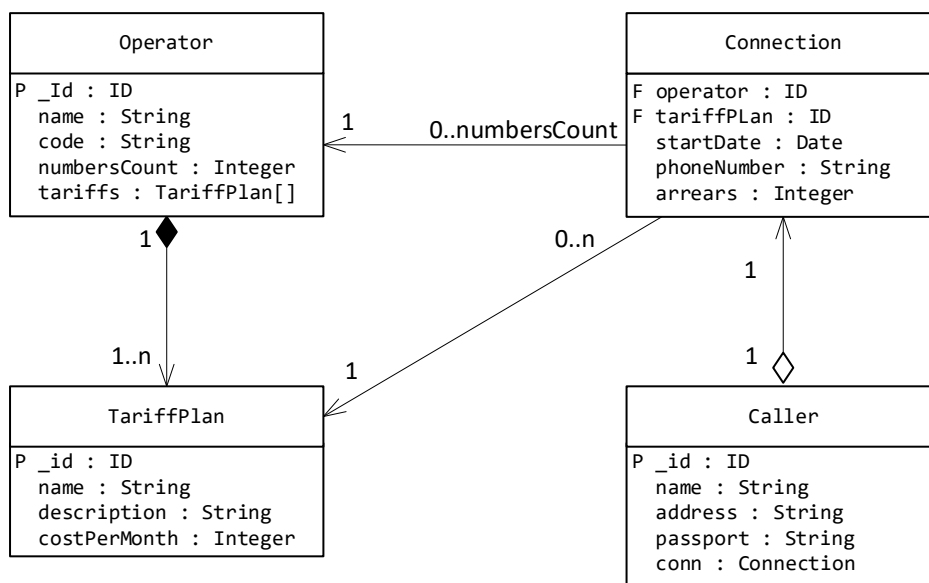


Рисунок 9 – схема взаимодействия сущностей

После проектирования я приступил к разработке приложения, реализующего ИС.

Были созданы 3 класса (рисунок 10).

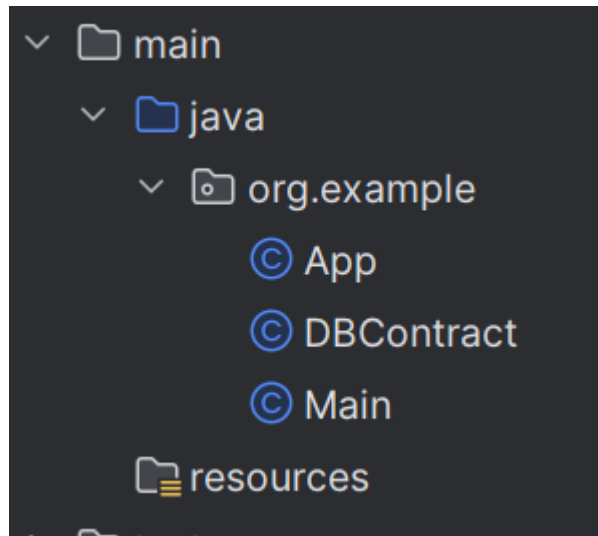


Рисунок 10 – Структура программы ИС

DBContract – класс содержащий в себе текстовые константы с названиями полей в документах и коллекций и текущую версию базы данных.

Main – класс запуска содержащий в себе статический метод запуска и обновления структуры базы данных (обновление нужно было для более удобного полного стирания а затем заполнения базы данных начальными значениями)

App – класс содержащий в себе интерфейс и основные методы работы с сущностями БД.

Теперь приведу тексты скриптов для создания и заполнения коллекций.

Механизм обновления БД (рисунок 11). Для контроля текущей версии базы данных была добавлена коллекция с настройками, содержащая номер текущей версии. При запуске программа проверяет, если документ настроек еще не создан, то база данных заполняется начальными данными (заполняет коллекцию операторов) и создает этот документ.


```

// коллекция настроек
MongoCollection<Document> collection =
    db.getCollection(DBContract.COLLECTION_PREFS);
// находим первый документ с id = 1
Document settingsDocument = collection
    .find(eq( fieldName: "_id", value: 1))
    .first();
// если он еще не был создан, значит БД пустая
if (settingsDocument == null) {
    // инициализация документа с номером версии бд
    settingsDocument = new Document()
        .append("_id", 1)
        .append(DBContract.FIELD_PREFS_VERSION, 0);
    collection.insertOne(settingsDocument);
}

{ // механизм обновления БД
    // проверяем самая ли последняя в бд версия
    int currentVersion = settingsDocument.getInteger(DBContract.FIELD_PREFS_VERSION);
    if (currentVersion != DBContract.VERSION) {

        // обновление/инициализирование БД
        updateDatabase(currentVersion, DBContract.VERSION, db);

        // обновляем документ настроек в программе (на всякий)
        settingsDocument.append(DBContract.FIELD_PREFS_VERSION, DBContract.VERSION);
        // обновляем документ настроек в БД
        collection.updateOne(eq( fieldName: "_id", value: 1),
            set(DBContract.FIELD_PREFS_VERSION, DBContract.VERSION)
        );
    }
}
}

```

Рисунок 11 – Программа инициализации и обновления БД

Главное меню, представляет из себя циклический выбор одного из трех пунктов. (Рисунки 12-13).

```

Hello and welcome!
::::::::::::::::::::: Главное меню :::::::::::::::::::::::
Доступные операции:
    0: Выход из программы;
    1: Регистрация абонента;
    2: Просмотр тарифных планов различных операторов;
    3: Покупка тарифа
1

```

Рисунок 12 – Вид интерфейса главного меню

```
// обработка ввода
switch (userInput) {
    case 0 -> {
        isExit = true;
        System.out.println("Завершение работы.");
    }
    case 1 -> registerCaller();
    case 2 -> viewPlans();
    case 3 -> bySomePlan();
    default -> System.out.println("Некорректное число");
}
```

Рисунок 13 – Код выбора одного из пунктов меню

Меню регистрации абонента (рисунок 14) позволяет добавить в систему нового пользователя.

```
1
::::::::::::::::: Регистрация абонента :::::::::::::::::::
Введите пожалуйста данные абонента:
  ФИО:Марчук Иван Сергеевич
  Адрес:Москва
  Номер паспорта:12345
  Если данные введены верно введите Y:y
ID пользователя:65745b0ecb435f43b83103ba
::::::::::::::::: Абонент успешно зарегистрирован! :::::::::::::::::::
```

Рисунок 14 – Интерфейс регистрации абонента

Поле Connection содержащееся в документе коллекции Caller изначально пустое (что видно из рисунка 15), его можно заполнить в меню «покупка тарифа».

```
// добавление пользователя в бд
MongoCollection<Document> collection = db.getCollection(DBContract.COLLECTION_CALLER);
Document document = new Document()
    .append(DBContract.FIELD_CALLER_NAME, name)
    .append(DBContract.FIELD_CALLER_ADDRESS, address)
    .append(DBContract.FIELD_CALLER_PASSPORT, passport)
    .append(DBContract.FIELD_CALLER_CONNECTION, null);
collection.insertOne(document);
ObjectId id = document.getObjectId(key: "_id");

System.out.println("ID пользователя:" + id);
System.out.println("::::::::::::::::: Абонент успешно зарегистрирован! :::::::::::::::::::");
```

Рисунок 15 – Программа добавления документа в коллекцию Caller

В compass можно посмотреть, что документ Caller действительно добавился (рисунок 16).

```
_id: ObjectId('65745b0ecb435f43b83103ba')
name: "Марчук Иван Сергеевич"
address: "Москва"
passport: "12345"
conn: null
```

Рисунок 16 – Документ в коллекции Caller в программе Compass

В меню «просмотр тарифных планов» (рисунок 17) можно посмотреть общий список операторов и сведения о каждом из них (код на рисунке 18), а при выборе определенного оператора выведутся его тарифы (код на рисунке 19).

```
2
::::::::::::::::: Просмотр тарифных планов :::::::::::::::::::
Список операторов доступных на данный момент:
- AT&T Код оператора: AT&T20CODE Количество номеров: 20 Количество тарифов: 3
- MTS Код оператора: MTS05CODE Количество номеров: 5 Количество тарифов: 2
- OnLINE Код оператора: OnLINE555CODE Количество номеров: 555 Количество тарифов: 3
Выберите нужного оператора введя его код (пустая строка - выход):
MTS05CODE
Тарифы данного оператора (Для покупки тарифа используется его название):
- Название тарифа:base Стоимость:100 Описание:Только звонки.
- Название тарифа:turbo Стоимость:500 Описание:Звонки + мобильный интернет с раздачей.
```

Рисунок 17 – Интерфейс просмотра операторов и тарифных планов

```
// вывод доступных операторов
MongoCollection<Document> operators = db.getCollection(DBContract.COLLECTION_OPERATOR);
operators.find().forEach((Consumer<Document>) document -> {
    System.out.printf(" - %s Код оператора: %s Количество номеров: %d Количество тарифов: %d \n",
        document.getString(DBContract.FIELD_OPERATOR_NAME),
        document.getString(DBContract.FIELD_OPERATOR_CODE),
        document.getInteger(DBContract.FIELD_OPERATOR_NUMBERS_COUNT),
        document.getList(DBContract.FIELD_OPERATOR_TARIFFS, Document.class).size());
});
```

Рисунок 18 – Программа вывода списка операторов

Получение списка тарифов происходит через коллекцию Java List, то есть весь массив документов «Tariff» из документа «Operator» выгружается в оперативную память. К сожалению по другому со вложенными массивами документов в MongoDB работать нельзя, а по тому вложенные коллекции не могут быть большими, чтобы не переполнялась оперативная память клиента.

```
String input = scanner.nextLine();
Document selectedOperator = operators.find(eq(DBContract.FIELD_OPERATOR_CODE, input)).first();
if (selectedOperator != null) {

    // получение списка тарифов
    Iterator<Document> tariffs = selectedOperator
        .getList(DBContract.FIELD_OPERATOR_TARIFFS, Document.class)
        .iterator();

    // вывод тарифов
    System.out.println("Тарифы данного оператора (Для покупки тарифа используется его название):");
    while (tariffs.hasNext()) {
        Document tariff = tariffs.next();

        System.out.printf(" - Название тарифа:%s  Стоимость:%d  Описание:%s\n",
            tariff.getString(DBContract.FIELD_TARIFF_NAME),
            tariff.getInteger(DBContract.FIELD_TARIFF_COST),
            tariff.getString(DBContract.FIELD_TARIFF_DESCRIPTION)
        );
    }
}
```

Рисунок 19 – Программа выбора оператора для отображения и вывода списка тарифов из этого оператора

```
_id: ObjectId('65736e079d4d104387e7e5e9')
name: "MTS"
code: "MTS05CODE"
numbersCount: 5
▼ tariffs: Array (2)
  ▼ 0: Object
    _id: ObjectId('65736e079d4d104387e7e5e3')
    name: "base"
    description: "Только звонки."
    costPerMonth: 100
  ► 1: Object
```

Рисунок 20 – Структура документа в коллекции Operator

Меню «покупка тарифа» позволяет найти добавленного «Caller» и посмотреть его документ с данными, где будет, в том числе, выведена информация о текущем подключении (рисунок 21).

Программа также позволяет выбрать оператора и тариф, чтобы добавить Абоненту (Caller), документ Connection (код приведен на рисунке 22).

Таким образом происходит подключение тарифа пользователю. Результат добавления представлен на рисунке 23.

```

3
::::::::::::: Покупка тарифного плана :::::::::::::::
Введите паспортные данные абонента:
12345
Абонент найден:
- Имя:Марчук Иван Сергеевич
- Адрес:Москва
- Паспорт:12345
- Тарифного плана пока нет!
Покупка/смена тарифного плана:
- AT&T Код оператора: AT&T20CODE
- MTS Код оператора: MTS05CODE
- OnLINE Код оператора: OnLINE555CODE
Введите код оператора:
MTS05CODE
Список тарифов оператора:
- base
- turbo
Введите название тарифа:
turbo
Подключение выполнено успешно

```

Рисунок 21 – Интерфейс меню подключения тарифного плана и просмотра данных пользователя

```

// создаем соединение
Document document = new Document()
    .append(DBContract.FIELD_CONNECTION_OPERATOR, selectedOperator.getObjectId( key: "_id"))
    .append(DBContract.FIELD_CONNECTION_TARIFF_PLAN, finDtariff)
    .append(DBContract.FIELD_CONNECTION_START_DATE,
        new SimpleDateFormat( pattern: "yyyy.MM.dd").format(new Date()))
    .append(DBContract.FIELD_CONNECTION_PHONE_NUMBER, "880000000")
    .append(DBContract.FIELD_CONNECTION_ARREARS, 0);

// добавляем CONNECTION в Caller
callers.updateOne(
    eq( fieldName: "_id", selectedCaller.getObjectId( key: "_id")),
    set(DBContract.FIELD_CALLER_CONNECTION, document));

System.out.println("Подключение выполнено успешно");

```

Рисунок 22 – Обновление поля connection (содержащим документ) в документе Caller

```
_id: ObjectId('65745b0ecb435f43b83103ba')
name: "Марчук Иван Сергеевич"
address: "Москва"
passport: "12345"
▼ conn: Object
  operator: ObjectId('65736e079d4d104387e7e5e9')
  tariffPlan: ObjectId('65736e079d4d104387e7e5e4')
  startDate: "2023.12.09"
  phoneNumber: "880000000"
  arrears: 0
```

Рисунок 23 – Структура документа Caller после добавления в неё документа Connection

Полный код ИС представлен в приложении А.

Вывод:

Я создал информационную систему на основе СУБД Mongo DB.
Получил опыт программирования на языке Java и работы с библиотекой java driver Mongo DB

Приложение А
Полный код информационной системы
Листов 10

Листинг класса DBContract:

```
package org.example;

public class DBContract {

    // константы с названиями полей и коллекций
    public static final String DB_NAME = "Communication3K";

    public static final String COLLECTION_PREFS = "Preferences";
    /*
     * P_Id : Integer
     * version : String
     */
    public static final String FIELD_PREFS_VERSION = "version";
    // Текущая версия БД
    public static final int VERSION = 2;

    public static final String COLLECTION_CALLER = "Caller";
    /*
     * P_id : ID
     * name : String
     * address : String
     * passport : String
     * conn : Connection
     */
    public static final String FIELD_CALLER_NAME = "name";
    public static final String FIELD_CALLER_ADDRESS = "address";
    public static final String FIELD_CALLER_PASSPORT = "passport";
    public static final String FIELD_CALLER_CONNECTION = "conn";

    public static final String COLLECTION_CONNECTION = "Connection";
    /*
     * P_id : ID
     * F operator : ID
     * F tariffPlan : ID
     * startDate : Date
     * phoneNumber : String
     * arrears : Integer
     */
    public static final String FIELD_CONNECTION_OPERATOR = "operator";
    public static final String FIELD_CONNECTION_TARIFF_PLAN = "tariffPlan";
    public static final String FIELD_CONNECTION_START_DATE = "startDate";
    public static final String FIELD_CONNECTION_PHONE_NUMBER = "phoneNumber";
    public static final String FIELD_CONNECTION_ARREARS = "arrears";

    public static final String COLLECTION_OPERATOR = "Operator";
    /*
     * P_Id : ID
     * name : String
     * code : String
     * numbersCount : Integer
     * tariffs : TariffPlan[]
     */
    public static final String FIELD_OPERATOR_NAME = "name";
    public static final String FIELD_OPERATOR_CODE = "code";
    public static final String FIELD_OPERATOR_NUMBERS_COUNT = "numbersCount";
    public static final String FIELD_OPERATOR_TARIFFS = "tariffs";
}
```



```

    public static final String COLLECTION_TARIFF = "TariffPlan";
    /*
    * P_id : ID
    * name : String
    * description : String
    * costPerMonth : Integer
    * */
    public static final String FIELD_TARIFF_NAME = "name";
    public static final String FIELD_TARIFF_DESCRIPTION = "description";
    public static final String FIELD_TARIFF_COST = "costPerMonth";

}

```

Листинг класса Main:

```

package org.example;
import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import org.bson.types.ObjectId;

import java.util.ArrayList;
import java.util.List;

import static com.mongodb.client.model.Filters.eq;
import static com.mongodb.client.model.Updates.set;

public class Main {

    public static void main(String[] args) {

        // подключаемся к СУБД
        System.out.println("Инициализация базы.");
        try (MongoClient myMongoClient = MongoClients.create(
            "mongodb://localhost:27017"//"mongodb://user:passwd@localhost:27017/?authSource=test"
        )) {
            // ждем когда MONGODB подключится
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            MongoDBDatabase db = myMongoClient.getDatabase(DBContract.DB_NAME);

            { // проверка бд на инициализированность

                // коллекция настроек
                MongoCollection<Document> collection =
                    db.getCollection(DBContract.COLLECTION_PREFS);
                // находим первый документ с id = 1
                Document settingsDocument = collection
                    .find(eq("_id", 1))
                    .first();
                // если он еще не был создан, значит БД пустая
                if (settingsDocument == null) {
                    // инициализация документа с номером версии бд
                    settingsDocument = new Document()
                        .append("_id", 1)

```

```

        .append(DBContract.FIELD_PREFS_VERSION, 0);
        collection.insertOne(settingsDocument);
    }

    { // механизм обновления БД
        // проверяем самая ли последняя в бд версия
        int currentVersion =
        settingsDocument.getInteger(DBContract.FIELD_PREFS_VERSION);
        if (currentVersion != DBContract.VERSION) {

            // обновление/инициализирование БД
            updateDatabase(currentVersion, DBContract.VERSION, db);

            // обновляем документ настроек в программе (на всякий)
            settingsDocument.append(DBContract.FIELD_PREFS_VERSION,
            DBContract.VERSION);

            // обновляем документ настроек в БД
            collection.updateOne(eq("_id", 1),
                set(DBContract.FIELD_PREFS_VERSION,
            DBContract.VERSION)
            );
        }
    }
}

System.out.println("Hello and welcome!");

// инициализация интерфейса пользователя
App app = new App(db);

// запуск главного меню
app.startMainMenu();

}

private static void updateDatabase(int oldVersion, int newVersion, MongoDBDatabase
db) {
    System.out.println("Update database from:" + oldVersion + "v to:" +
newVersion + "v");

    // пустая БД
    if (oldVersion == 0) {
        // заполняем коллекции операторов и тарифов

        MongoCollection<Document> operators =
            db.getCollection(DBContract.COLLECTION_OPERATOR);
        // чистка всего
        operators.drop();

        // создание списка операторов
        List<Document> operatorsDocs = new ArrayList<>();
        {
            // список тарифов конкретного оператора
            List<Document> tariffDocsATNT = new ArrayList<>();
            tariffDocsATNT.add(new Document()
                .append("_id", new ObjectId())
                .append(DBContract.FIELD_TARIFF_NAME, "smallTariff")
                .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Минимум
функций, только необходимое.")
                .append(DBContract.FIELD_TARIFF_COST, 100));
        }
    }
}

```

```

tariffDocsATNT.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "mediumTariff")
    .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Как хлебушек,
но с маслом.")
    .append(DBContract.FIELD_TARIFF_COST, 200));
tariffDocsATNT.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "SuperTariff")
    .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Любой каприз за
ваши деньги.")
    .append(DBContract.FIELD_TARIFF_COST, 500));
// создание оператора
operatorsDocs.add(new Document()
    .append(DBContract.FIELD_OPERATOR_NAME, "AT&T")
    .append(DBContract.FIELD_OPERATOR_CODE, "AT&T20CODE")
    .append(DBContract.FIELD_OPERATOR_NUMBERS_COUNT, 20)
    .append(DBContract.FIELD_OPERATOR_TARIFFS, tariffDocsATNT)
);
}
{
// список тарифов конкретного оператора
List<Document> tariffDocsMTS = new ArrayList<>();
tariffDocsMTS.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "base")
    .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Только
звонки.")
    .append(DBContract.FIELD_TARIFF_COST, 100));
tariffDocsMTS.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "turbo")
    .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Звонки +
мобильный интернет с раздачей.")
    .append(DBContract.FIELD_TARIFF_COST, 500));
// создание оператора
operatorsDocs.add(new Document()
    .append(DBContract.FIELD_OPERATOR_NAME, "MTS")
    .append(DBContract.FIELD_OPERATOR_CODE, "MTS05CODE")
    .append(DBContract.FIELD_OPERATOR_NUMBERS_COUNT, 5)
    .append(DBContract.FIELD_OPERATOR_TARIFFS, tariffDocsMTS)
);
}
{
// список тарифов конкретного оператора
List<Document> tariffDocsOnLine = new ArrayList<>();
tariffDocsOnLine.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "asket")
    .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Минимум
функций, только необходимое.")
    .append(DBContract.FIELD_TARIFF_COST, 250));
tariffDocsOnLine.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "middle man")
    .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Средний такой
тариф")
    .append(DBContract.FIELD_TARIFF_COST, 300));
tariffDocsOnLine.add(new Document()
    .append("_id", new ObjectId())
    .append(DBContract.FIELD_TARIFF_NAME, "gigachad")

```

```

        .append(DBContract.FIELD_TARIFF_DESCRIPTION, "Вы не найдете
тарифа лучше!")
        .append(DBContract.FIELD_TARIFF_COST, 400));
// создание оператора
operatorsDocs.add(new Document()
    .append(DBContract.FIELD_OPERATOR_NAME, "OnLINE")
    .append(DBContract.FIELD_OPERATOR_CODE, "OnLINE555CODE")
    .append(DBContract.FIELD_OPERATOR_NUMBERS_COUNT, 555)
    .append(DBContract.FIELD_OPERATOR_TARIFFS, tariffDocsOnLine)
);
}
operators.insertMany(operatorsDocs);
}
}
}

```

Листинг класса App:

```

package org.example;

import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import org.bson.types.ObjectId;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;
import java.util.function.Consumer;

import static com.mongodb.client.model.Filters.eq;
import static com.mongodb.client.model.Updates.set;

public class App {

    private final MongoDatabase db;
    private final Scanner scanner;

    // инициализация интерфейса
    public App(MongoDatabase db) {
        this.db = db;
        scanner = new Scanner(System.in);
    }

    // запуск работы интерфейса
    void startMainMenu() {

        // главный интерфейсный цикл
        boolean isExit = false;
        while (!isExit) {

            System.out.println("""
            ::::::::::::::: Главное меню :::::::::::::::
            Доступные операции:
            0: Выход из программы;
            1: Регистрация абонента;
            2: Просмотр тарифных планов различных операторов;
            3: Покупка тарифа""");

```

```

        // ввод пользователя
        int userInput = -1;
        try {
            userInput = Integer.parseInt(scanner.nextLine());
        } catch (NumberFormatException ignored) {
        }

        // обработка ввода
        switch (userInput) {
            case 0 -> {
                isExit = true;
                System.out.println("Завершение работы.");
            }
            case 1 -> registerCaller();
            case 2 -> viewPlans();
            case 3 -> bySomePlan();
            default -> System.out.println("Некорректное число");
        }
        scanner.nextLine();
    }
}

// Регистрация абонента
private void registerCaller() {
    System.out.println("::::::::::::::::: Регистрация абонента ::::::::::::::::::");

    System.out.println("Введите пожалуйста данные абонента:");
    System.out.print("  ФИО:");
    String name = scanner.nextLine();
    System.out.print("  Адрес:");
    String address = scanner.nextLine();
    System.out.print("  Номер паспорта:");
    String passport = scanner.nextLine();

    System.out.print("  Если данные введены верно введите Y:");
    String confirm = scanner.nextLine();
    if (confirm.charAt(0) == 'Y' || confirm.charAt(0) == 'y' ||
        confirm.charAt(0) == 'H' || confirm.charAt(0) == 'h') {

        // добавление пользователя в бд
        MongoCollection<Document> collection =
            db.getCollection(DBContract.COLLECTION_CALLER);
        Document document = new Document()
            .append(DBContract.FIELD_CALLER_NAME, name)
            .append(DBContract.FIELD_CALLER_ADDRESS, address)
            .append(DBContract.FIELD_CALLER_PASSPORT, passport)
            .append(DBContract.FIELD_CALLER_CONNECTION, null);
        collection.insertOne(document);
        ObjectId id = document.getObjectId("_id");

        System.out.println("ID пользователя: " + id);
        System.out.println("::::::::::::: Абонент успешно зарегистрирован!
:::::::::::::");
    } else {

        System.out.println("::::::::::::::::::::::::: Отмена операции
:::::::::::::::::::::::::");
    }
}

// Просмотр тарифных планов различных операторов

```

```

        private void viewPlans() {
            System.out.println("::::::::::::: Просмотр тарифных планов
:::::::::::::");
            System.out.println("Список операторов доступных на данный момент:");

            // вывод доступных операторов
            MongoClient<Document> operators =
            db.getCollection(DBContract.COLLECTION_OPERATOR);
            operators.find().forEach((Consumer<Document>) document -> {
                System.out.printf(" - %s Код оператора: %s Количество номеров: %d
Количество тарифов: %d \n",
                    document.getString(DBContract.FIELD_OPERATOR_NAME),
                    document.getString(DBContract.FIELD_OPERATOR_CODE),
                    document.getInteger(DBContract.FIELD_OPERATOR_NUMBERS_COUNT),
                    document.getList(DBContract.FIELD_OPERATOR_TARIFFS,
Document.class).size());
            });

            // выбор оператора
            System.out.println("Выберите нужного оператора введя его код (пустая строка -
выход):");
            String input = scanner.nextLine();
            Document selectedOperator = operators.find(eq(DBContract.FIELD_OPERATOR_CODE,
input)).first();
            if (selectedOperator != null) {

                // получение списка тарифов
                Iterator<Document> tariffs = selectedOperator
                    .getList(DBContract.FIELD_OPERATOR_TARIFFS, Document.class)
                    .iterator();

                // вывод тарифов
                System.out.println("Тарифы данного оператора (Для покупки тарифа
используется его название):");
                while (tariffs.hasNext()) {
                    Document tariff = tariffs.next();

                    System.out.printf(" - Название тарифа:%s  Стоимость:%d
Описание:%s\n",
                        tariff.getString(DBContract.FIELD_TARIFF_NAME),
                        tariff.getInteger(DBContract.FIELD_TARIFF_COST),
                        tariff.getString(DBContract.FIELD_TARIFF_DESCRIPTION)
                    );
                }
            }
        }

        // Покупка абонентом тарифа
        private void bySomePlan() {
            System.out.println("::::::::::::: Покупка тарифного плана
:::::::::::::");

            MongoClient<Document> callers =
            db.getCollection(DBContract.COLLECTION_CALLER);
            // Здесь можно вывести список пользователей... потом

            System.out.println("Введите паспортные данные абонента:");
            Document selectedCaller = callers.find(eq(DBContract.FIELD_CALLER_PASSPORT,
scanner.nextLine())).first();
            if (selectedCaller != null) {
                System.out.println("Абонент найден:");
            }
        }
    }
}

```

```

        System.out.println(" - Имя:" +
selectedCaller.getString(DBContract.FIELD_CALLER_NAME));
        System.out.println(" - Адрес:" +
selectedCaller.getString(DBContract.FIELD_CALLER_ADDRESS));
        System.out.println(" - Паспорт:" +
selectedCaller.getString(DBContract.FIELD_CALLER_PASSPORT));

        // текущее соединение
        Document connection = (Document)
selectedCaller.get(DBContract.FIELD_CALLER_CONNECTION);
        if (connection == null) {
            System.out.println(" - Тарифного плана пока нет!");
        } else {

            System.out.println(" - Подключен тарифный план:");

            // загружаем данные об операторе и тарифе
            MongoCollection<Document> operators =
db.getCollection(DBContract.COLLECTION_OPERATOR);
            Document operator =
                operators.find(eq("_id",
connection.getObjectId(DBContract.FIELD_CONNECTION_OPERATOR))).first();

            if(operator != null) {
                String tariffName = null;
                Iterator<Document> tariffsIterator =
                    operator.getList(DBContract.FIELD_OPERATOR_TARIFFS,
Document.class).iterator();
                while (tariffsIterator.hasNext() && tariffName == null) {
                    Document document = tariffsIterator.next();
                    if
(document.getObjectId("_id").equals(connection.getObjectId(DBContract.FIELD_CONNECTIO
N_TARIFF_PLAN))) {
                        tariffName =
document.getString(DBContract.FIELD_TARIFF_NAME);
                    }
                }
                System.out.println("    + Оператор:" +
                    ((operator != null) ?

(operator.getString(DBContract.FIELD_OPERATOR_NAME)) :
                    ("Отсутствует")));
                System.out.println("    + Тарифный план:" + tariffName);
            }else{
                System.out.println("    + Оператор:Отсутствует \n    + Тарифный
план:Отсутствует");
            }
            System.out.println("    + Дата подключения:" +
connection.getString(DBContract.FIELD_CONNECTION_START_DATE));
            System.out.println("    + Телефон:" +
connection.getString(DBContract.FIELD_CONNECTION_PHONE_NUMBER));
            System.out.println("    + Задолженность:" +
connection.getInteger(DBContract.FIELD_CONNECTION_ARREARS));
        }

        System.out.println("Покупка/смена тарифного плана: ");

        // вывод доступных операторов
        MongoCollection<Document> operators =
db.getCollection(DBContract.COLLECTION_OPERATOR);
        operators.find().forEach((Consumer<Document>) document ->
            System.out.printf(" - %s Код оператора: %s \n",

```

```

        document.getString(DBContract.FIELD_OPERATOR_NAME),
        document.getString(DBContract.FIELD_OPERATOR_CODE)));

// выбор оператора
System.out.println("Введите код оператора: ");
Document selectedOperator =
operators.find(eq(DBContract.FIELD_OPERATOR_CODE, scanner.nextLine())).first();
if (selectedOperator == null) {
    System.out.println("Оператор не найден.");
} else {
    // Правда код с List не будет работать на большом количестве данных
    // (он загружает сразу всю коллекцию в отличие от
курсора(FindIterable))
    // по этому тарифов у оператора не может быть больше например 100

    // получение списка тарифов
    List<Document> tariffs = selectedOperator
        .getList(DBContract.FIELD_OPERATOR_TARIFFS, Document.class);

    { // вывод тарифов
        Iterator<Document> tariffsI = tariffs.iterator();
        System.out.println("Список тарифов оператора:");
        while (tariffsI.hasNext()) {
            Document tariff = tariffsI.next();
            System.out.println(" - " +
tariff.getString(DBContract.FIELD_TARIFF_NAME));
        }
    }

    System.out.println("Введите название тарифа: ");
    { // выбор тарифа
        String input = scanner.nextLine();

        ObjectId finDtariff = null;

        // проходимся по коллекции тарифов
        Iterator<Document> tariffsI = tariffs.iterator();
        while (tariffsI.hasNext() && finDtariff == null) {
            // если у этого тарифа нужно имя
            Document tariffUnit = tariffsI.next();
            if
(input.equals(tariffUnit.getString(DBContract.FIELD_TARIFF_NAME))) {
                finDtariff = tariffUnit.getObjectId("_id");
            }
        }
        // Если такой тариф есть, добавляем CONNECTION
        if (finDtariff != null) {

            // создаем соединение
            Document document = new Document()
                .append(DBContract.FIELD_CONNECTION_OPERATOR,
selectedOperator.getObjectId("_id"))
                .append(DBContract.FIELD_CONNECTION_TARIFF_PLAN,
finDtariff)
                .append(DBContract.FIELD_CONNECTION_START_DATE,
                    new SimpleDateFormat("yyyy.MM.dd").format(new
Date()))
                .append(DBContract.FIELD_CONNECTION_PHONE_NUMBER,
"880000000")
                .append(DBContract.FIELD_CONNECTION_ARREARS, 0);

            // добавляем CONNECTION в Caller

```



```
        callers.updateOne(
            eq("_id", selectedCaller.getObjectId("_id")),
            set(DBContract.FIELD_CALLER_CONNECTION, document));

        System.out.println("Подключение выполнено успешно");
    }
}
}
```