Санкт-Петербургский Государственный Политехнический Университет
Кафедра Компьютерных Систем и Программных Технологий

ОТЧЕТ

по лабораторной работе

«Разработка IMAP-сервера»

Дисциплина: Сети ЭВМ и телекоммуникации

Работу выполнил студент гр. 43501/3: Муравьёв Ф.Э.

Преподаватель: Вылегжанина К.Д.

Санкт-Петербург
2016

1. Техническое задание

Разработать приложение для операционных систем семейства Windows, обеспечивающее функции сервера протокола IMAP4.

1.1. Основные возможности

Приложение должно реализовывать следующие функции для подключаемых клиентов:

1. Подключение к указанному серверу по IP-адресу или доменному имени;
2. Получение состояния ящика (количество новых писем, общее количество писем);
3. Вход в конкретную папку сервера для работы;
4. Получение списка заголовков всех новых писем сервера без предварительной загрузки;
5. Загрузка всех новых или конкретных выбранных писем с сервера;
6. Работа с папками и подпапками.

По согласованию с преподавателем приложение может выполнять следующие функции протокола:

1. Создавать/Изменять/Удалять папки;
2. Загружать письма на сервер;
3. Выполнять проверку совместимости сервера;
4. Подключение к серверу в режиме Read-Only;
5. Возможность копирования писем между папками;
6. Смена флагов письма;

1.2. Поддерживаемые команды

Разработанное приложение должно реализовать следующие команды протокола IMAP ver.4:

- LOGIN – передача серверу идентификационной информации и пароля пользователя;
- LIST – получение списка папок почтового ящика;
- STATUS – получение информации о конкретной папке;
- SELECT – вход в папку;
- FETCH – загрузка сообщения;
- STORE – установка флагов сообщения;
- LOGOUT – выход с сервера;
- CAPABILITY – получение информации о совместимости;
- NOOP – обновление информации о почтовом ящике;
- CREATE – создание папки;
- SUBSCRIBE – подписаться на папку;
- LSUB – список подписанных папок;
- COPY – копирование писем;
- UID – получение уникальных идентификаторов писем.

1.3. Настройка приложения

Разработанное приложение должно предоставлять пользователю настройку следующих параметров:

1. IP-адрес или доменное имя почтового сервера;
2. Номер пора сервера (по умолчанию - 143)

1.4. Методика тестирования

Для тестирования приложения следует использовать почтовые клиенты, имеющиеся в лаборатории (Mozilla Thunderbird, MS Outlook Express, The Bat!, Netscape Messenger).

2. Теоретические сведения

IMAP - протокол прикладного уровня для доступа к электронной почте. Базируется на транспортном протоколе TCP и использует порт 143.

IMAP предоставляет пользователю обширные возможности для работы с почтовыми ящиками, находящимися на центральном сервере. Почтовая программа, использующая этот протокол, получает доступ к хранилищу корреспонденции на сервере так, как будто эта корреспонденция расположена на компьютере получателя. Электронными письмами можно манипулировать с компьютера пользователя (клиента) без постоянной пересылки с сервера и обратно файлов с полным содержанием писем.

Сервер IMAP 4 находится в одном из четырёх состояний (рис. 1.).
Большинство команд можно использовать только в определенных состояниях:

- В состоянии *без аутентификации* клиент должен предоставить имя и пароль, прежде чем ему станет доступно большинство команд. Переход в это состояние производится при установлении соединения без предварительной аутентификации;
- В состоянии *аутентификации* клиент идентифицирован и должен выбрать почтовый ящик, после чего ему станут доступны команды для работы с сообщениями. Переход в это состояние происходит при установлении соединение с предварительной аутентификацией, когда выданы все необходимые идентификационные данные или при ошибочном выборе почтового ящика;
- В состояние *выбора* система попадает, когда успешно осуществлен выбор почтового ящика;
- В состояние *выхода* система попадает при прерывании соединения в результате запроса клиента или вследствие независимого решения сервера.

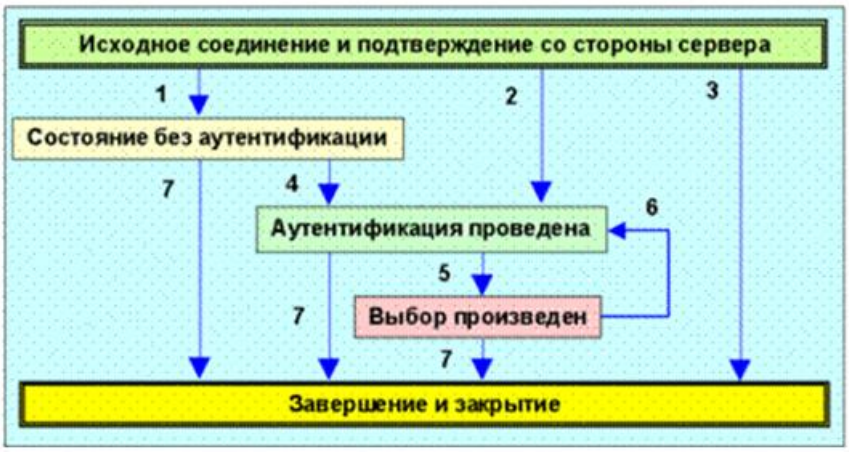


Рис 1. Состояния сервера

- (1) Соединение без предварительной аутентификации
- (2) Соединение с предварительной аутентификацией
- (3) Соединение отвергнуто
- (4) Успешное завершение команды LOGIN или AUTHENTICATE
- (5) Успешное завершение команды SELECT или EXAMINE
- (6) Выполнение команды CLOSE или неудачная команда SELECT или EXAMINE
- (7) Выполнение команды LOGOUT, закрытие сервера, или прерывание соединения

Отклики сервера делятся на три типа — информация о состоянии, данные и запросы на продолжение команд.

Клиент ДОЛЖЕН сохранять постоянную готовность к получению откликов от сервера.

Статусные отклики могут быть отмеченными (tagged) или неотмеченными (untagged). Отмеченные отклики показывают результат (OK, NO, BAD) выполнения команды клиента и включают тег, соответствующий команде.

Все данные и некоторые статусные отклики передаются без тегов. Непомеченные отклики идентифицируются маркером "*" взамен тега. Непомеченные статусные отклики содержат приветствия сервера, или данные о состоянии, которые не показывают завершения команд (например, уведомление об отключении сервера — shutdown alert). В силу исторических причин неотмеченные статусные отклики сервера называют также unsolicited data (представленные без запроса данные), хотя, строго говоря, они не являются таковыми.

Некоторые данные от сервера ДОЛЖНЫ записываться клиентом — такие случаи явно указываются в описании данных. Данные этого типа содержат критически важную информацию, которая влияет на интерпретацию последующих команд и откликов (например, обновлений о создании или удалении сообщений).

Другие типы данных следует сохранять для использования в будущем; если клиенту не нужно записывать эти данные или смысл записи данных неясен (например, отклик SEARCH при отсутствии выполняемых команд SEARCH), такие данные следует игнорировать.

Непомеченные данные по инициативе сервера могут передаваться, когда соединение IMAP находится в состоянии selected. В этом состоянии сервер проверяет почтовый ящик на предмет появления в нем новых сообщений в результате выполнения команд. Обычно

это является частью выполнения всех команд, следовательно, команда NOOP также инициирует проверку наличия новых сообщений. Если такие сообщения найдены, сервер передает непомеченные отклики EXISTS и RECENT, отражающие новый размер почтового ящика. Серверам, поддерживающим множественный доступ к почтовому ящику, следует также передавать подходящие непомеченные отклики FETCH и EXPUNGE, если другой агент изменяет состояние любого из флагов сообщений или удаляет какое-либо сообщение.

Запросы на продолжение команд используют маркер "+" взамен тега. Такие отклики передаются сервером для индикации восприятия сервером неполной команды клиента и готовности к получению оставшейся части команды.

3. Структура проекта

Проект состоит из двух основных частей:

1. Программная часть.
2. База данных.

3.1Программная часть

Программная часть представляет собой программу написанную на языке Java, реализующая все функции, перечисленные в техническом задании. UML-диаграмма представлена на рис.2.

Часть команд была реализована в виде неподдерживаемых команд сервера. Это значит, что получая такую команду, сервер отвечал “BAD”. Сделано это для сохранения целостности процедуры общения клиент-сервер. К таким командам относятся:

- Subscribe;
- Lsub.

Точка входа в приложение – класс ServerControl, который запускает работу сервера. В main(String[] args) происходит инициализация сервера. Класс IncomingServer – основной класс, который представляет собой как таковой сервер.

Для работы с базой данных есть класс DB, которые отвечает за выполнение SQL запросов.

Сервер умеет сохранять письма в формате eml. В этом ему помогает класс MimeMessage, который удобно представляет письмо в виде одного объекта. Локально хранятся письма клиента, как только клиент подключается к серверу, сразу же получает все письма. Письма могут маркироваться флагами и сервер учитывает это.

В пакете commands реализованы все команды, которые могут обрабатываться сервером. Как только отклик готов, сервер отправляет его клиенту. Каждая команда наследует класс IncomingServer, это удобно, но и есть недостаток в том, что все объекты имеют доступ к методу создающий их же объект. При ошибке возможна рекурсия.

Полный код программы представлен в приложении 1.

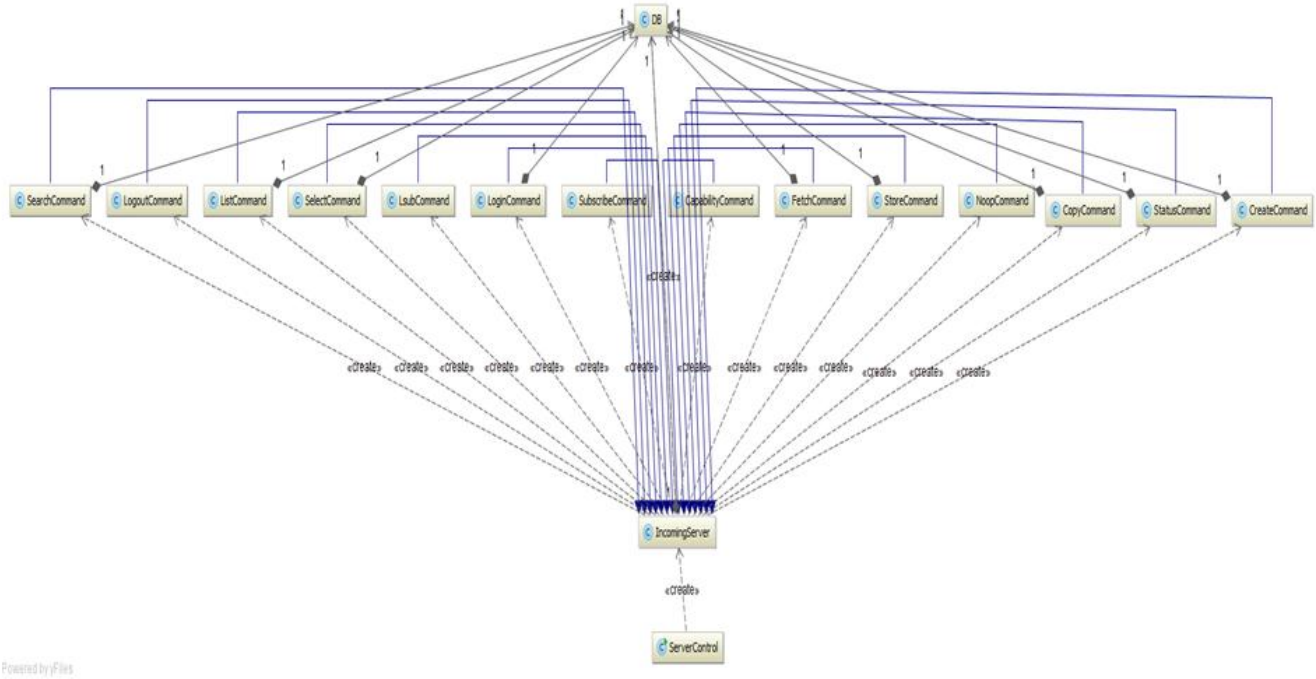


рис 2. UML диаграмма проекта

3.2База данных

База данных в данном проекте используется для хранения каталога пользователей, а так же списка их писем и флагов. Сами письма в базе не хранятся. База данных реализована при помощи СУБД firebird. Структура базы представлена на рисунке 3.

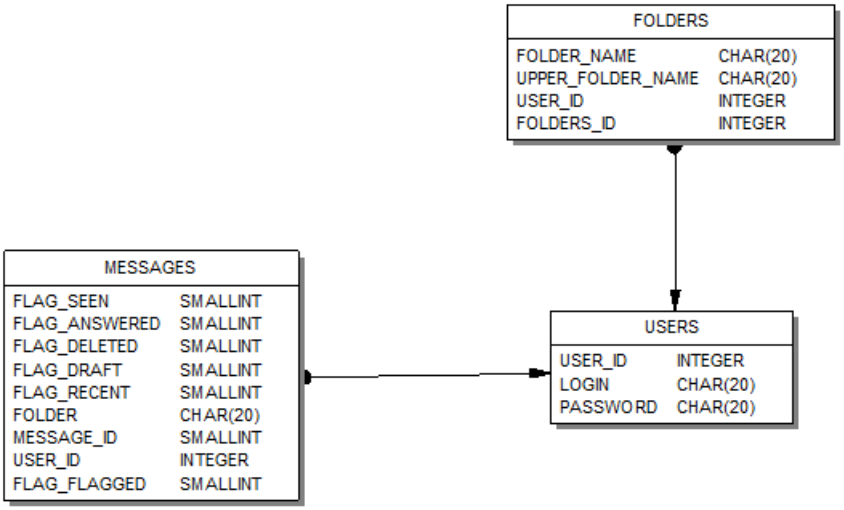


Рис.3. Структура базы данных

4. Тестирование проекта

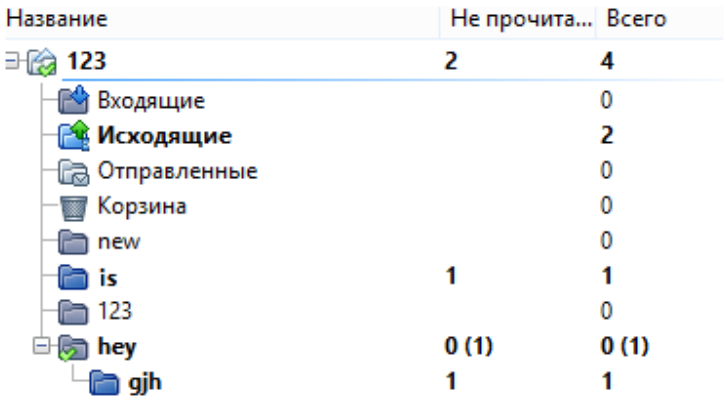
Основным почтовым клиентом для тестирования был выбран The Bat! По сравнению с другими программами его проще всего настраивать и удобный интерфейс.

При подключении к серверу клиент совершает сразу несколько действий по получению новой почты (Листинг 1).

Листинг 1. Подключение клиента

```
Server at 143 running
Waiting for a client...
Client connected
Clent says: 00001 CAPABILITY
IncomingServer says: * CAPABILITY IMAP4rev1
IncomingServer says: 00001 OK CAPABILITY completed
Clent says: 00002 LOGIN "log" "pas"
IncomingServer says: 00002 OK LOGIN completed
Clent says: 00003 SELECT "INBOX"
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 1] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00003 OK [READ-WRITE] SELECT Completed
Clent says: 00004 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00004 OK SEARCH completed
Clent says: 00005 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00005 OK SEARCH completed
Clent says: 00006 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00006 OK SEARCH completed
Clent says: 00007 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00007 OK SEARCH completed
Clent says: 00008 STATUS "new" (MESSAGES UNSEEN RECENT UIDNEXT UIDVALIDITY)
IncomingServer says: 00008 OK STATUS completed
IncomingServer says: * STATUS new (UIDNEXT 5 MESSAGES 1 UNSEEN 1 RECENT 0)
Clent says: 00009 STATUS "is" (MESSAGES UNSEEN RECENT UIDNEXT UIDVALIDITY)
IncomingServer says: 00009 OK STATUS completed
IncomingServer says: * STATUS is (UIDNEXT 5 MESSAGES 1 UNSEEN 1 RECENT 1)
Clent says: 00010 STATUS "123" (MESSAGES UNSEEN RECENT UIDNEXT UIDVALIDITY)
IncomingServer says: 00010 OK STATUS completed
IncomingServer says: * STATUS 123 (UIDNEXT 5 MESSAGES 0 UNSEEN 0 RECENT 0)
Clent says: 00011 STATUS "hey" (MESSAGES UNSEEN RECENT UIDNEXT UIDVALIDITY)
IncomingServer says: 00011 OK STATUS completed
IncomingServer says: * STATUS hey (UIDNEXT 5 MESSAGES 0 UNSEEN 0 RECENT 0)
Clent says: 00012 STATUS "hey|gjh" (MESSAGES UNSEEN RECENT UIDNEXT UIDVALIDITY)
IncomingServer says: 00012 OK STATUS completed
IncomingServer says: * STATUS hey|gjh (UIDNEXT 5 MESSAGES 1 UNSEEN 1 RECENT 1)
Clent says: 00013 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00013 OK SEARCH completed
Clent says: 00014 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00014 OK SEARCH completed
Clent says: 00015 NOOP
IncomingServer says: 00015 OK NOOP completed
```

В результате, клиент получает список всех папок и сообщений в них (рис. 4).



При изменении количества писем перемещении их между папкам, изменении флагов, все отображается верно (Листинг 2, 3).

Листинг 2. Смена флага

```
Clent says: 00052 UID STORE 1 +FLAGS (\Flagged)
IncomingServer says: * STORE 1 FLAGS (\Flagged)
IncomingServer says: 00052 OK STORE completed
Clent says: 00053 SELECT "INBOX"
SELECT MESSAGE_ID AS "MESSAGE_ID" FROM MESSAGES WHERE USER_ID = '0' AND FLAG_SEEN = 0
AND FOLDER = 'INBOX' ORDER BY MESSAGE_ID ASCENDING
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 9] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00053 OK [READ-WRITE] SELECT Completed
```

Листинг 3. Копирование письма

```
Clent says: 00073 UID COPY 1 "is"
start = 15 end = 15
UID = 1
UPDATE MESSAGES SET FOLDER = 'is' WHERE MESSAGE_ID = '1'
IncomingServer says: 00073 COPY 1 "is"
IncomingServer says: 00073 OK COPY completed
Clent says: 00074 STATUS "is" (MESSAGES UNSEEN RECENT UIDNEXT UIDVALIDITY)
IncomingServer says: 00074 OK STATUS completed
```

Но при создании новой папки происходит зацикливание запросов (Листинг 4). Клиент до бесконечности запрашивает список писем. Вероятнее всего это связанно с тем, что отправляется ответ в каком-то неверном формате, возможно неверная кодировка. Либо, это связанно с тем, что при запросе FETCH с неверными аргументами нужно отвечать BAD а не NO.

Не смотря на вышеописанную проблему, новые папки создаются и корректно работают после перезапуска сервера.

Листинг 4. Создание новой папки

```
Clent says: 00122 CREATE "hey|3232"
IncomingServer says: 00122 OK CREATE completed
Clent says: 00123 SUBSCRIBE "hey|3232"
IncomingServer says: 00123 BAD SUBSCRIBE illegal command
Clent says: 00124 LIST "" "hey|3232"
IncomingServer says: * LIST (\NoInferiors) "|" hey|3232
IncomingServer says: 00124 OK LIST completed
Clent says: 00125 SELECT "hey|3232"
```

```
IncomingServer says: * 0 EXISTS
* 0 RECENT
* OK [UNSEEN 0]
* OK [UIDVALIDITY 22] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00125 OK [READ-WRITE] SELECT Completed
Clent says: 00126 UID FETCH 1:* (UID RFC822.SIZE INTERNALDATE FLAGS ENVELOPE
BODYSTRUCTURE)
IncomingServer says: 00126 No FETCH no one message
Clent says: 00127 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00127 OK SEARCH completed
Clent says: 00128 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00128 OK SEARCH completed
Clent says: 00129 SELECT "INBOX"
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 23] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00129 OK [READ-WRITE] SELECT Completed
Clent says: 00130 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00130 OK SEARCH completed
Clent says: 00131 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00131 OK SEARCH completed
Clent says: 00132 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00132 OK SEARCH completed
Clent says: 00133 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00133 OK SEARCH completed
Clent says: 00134 SELECT "hey|3232"
IncomingServer says: * 0 EXISTS
* 0 RECENT
* OK [UNSEEN 0]
* OK [UIDVALIDITY 24] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00134 OK [READ-WRITE] SELECT Completed
Clent says: 00135 UID FETCH 1:* (UID RFC822.SIZE INTERNALDATE FLAGS ENVELOPE
BODYSTRUCTURE)
IncomingServer says: 00135 No FETCH no one message
Clent says: 00136 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00136 OK SEARCH completed
Clent says: 00137 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00137 OK SEARCH completed
Clent says: 00138 SELECT "INBOX"
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 25] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00138 OK [READ-WRITE] SELECT Completed
Clent says: 00139 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00139 OK SEARCH completed
```

```
Clent says: 00140 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00140 OK SEARCH completed
Clent says: 00141 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00141 OK SEARCH completed
Clent says: 00142 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00142 OK SEARCH completed
Clent says: 00143 SELECT "hey|3232"
IncomingServer says: * 0 EXISTS
* 0 RECENT
* OK [UNSEEN 0]
* OK [UIDVALIDITY 26] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00143 OK [READ-WRITE] SELECT Completed
Clent says: 00144 UID FETCH 1:* (UID RFC822.SIZE INTERNALDATE FLAGS ENVELOPE
BODYSTRUCTURE)
IncomingServer says: 00144 No FETCH no one message
Clent says: 00145 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00145 OK SEARCH completed
Clent says: 00146 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00146 OK SEARCH completed
Clent says: 00147 SELECT "INBOX"
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 27] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00147 OK [READ-WRITE] SELECT Completed
Clent says: 00148 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00148 OK SEARCH completed
Clent says: 00149 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00149 OK SEARCH completed
Clent says: 00150 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00150 OK SEARCH completed
Clent says: 00151 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00151 OK SEARCH completed
Clent says: 00152 SELECT "hey|3232"
IncomingServer says: * 0 EXISTS
* 0 RECENT
* OK [UNSEEN 0]
* OK [UIDVALIDITY 28] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00152 OK [READ-WRITE] SELECT Completed
Clent says: 00153 UID FETCH 1:* (UID RFC822.SIZE INTERNALDATE FLAGS ENVELOPE
BODYSTRUCTURE)
IncomingServer says: 00153 No FETCH no one message
Clent says: 00154 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00154 OK SEARCH completed
Clent says: 00155 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00155 OK SEARCH completed
Clent says: 00156 SELECT "INBOX"
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 29] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00156 OK [READ-WRITE] SELECT Completed
```

```
Clent says: 00157 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00157 OK SEARCH completed
Clent says: 00158 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00158 OK SEARCH completed
Clent says: 00159 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00159 OK SEARCH completed
Clent says: 00160 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00160 OK SEARCH completed
Clent says: 00161 SELECT "hey|3232"
IncomingServer says: * 0 EXISTS
* 0 RECENT
* OK [UNSEEN 0]
* OK [UIDVALIDITY 30] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00161 OK [READ-WRITE] SELECT Completed
Clent says: 00162 UID FETCH 1:* (UID RFC822.SIZE INTERNALDATE FLAGS ENVELOPE
BODYSTRUCTURE)
IncomingServer says: 00162 No FETCH no one message
Clent says: 00163 UID SEARCH 1:* DELETED
IncomingServer says: * SEARCH
IncomingServer says: 00163 OK SEARCH completed
Clent says: 00164 UID SEARCH 1:* UNSEEN
IncomingServer says: * SEARCH
IncomingServer says: 00164 OK SEARCH completed
Clent says: 00165 SELECT "INBOX"
IncomingServer says: * 1 EXISTS
* 1 RECENT
* OK [UNSEEN 1] Message 4 is first unseen
* OK [UIDVALIDITY 31] UIDs valid
* FLAGS (\Answered \Recent \Deleted \Seen \Draft \Flagged)
* OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
IncomingServer says: 00165 OK [READ-WRITE] SELECT Completed
                                                ...
```

Основные команды были так же приведены в листинге 4.

При тестировании в других программах наблюдались различные проблемы. Например:

- Microsoft Outlook нельзя использовать, пока не настроен SMTP сервер;
- Mozilla Thunderbird использует довольно не обычный синтаксис, с буквами различных регистров.

5. Выводы

В следствии анализа логирования можно сказать что сервер работает корректно. Тестируемый клиент The Bat!. Существует проблема тестирования в том, что некоторые клиенты не предоставляют возможности авторизации без шифрования. По этому причине тестирование было произведено на одном клиенте. Клиент ни разу не сообщил об ошибке и вел себя корректно.

Данная реализация имеет некоторые недостатки:

- Зацикливание при создании новой папки.
- Чувствительность к регистру команд.
- Недочеты архитектуры.

6. Приложение

Класс ServerControl

```
package Server;
public class ServerControl
{
    /**
     * Точка входа в программу
     * @param args
     */
    public static void main(String[] args)
    {
        IncomingServer Incoming = new IncomingServer(8000);
        Incoming.run();
    }
}
```

Класс IncomingServer

```
package Server;

import Commands.*;
import DBHandler.DB;

import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.internet.MimeMessage;
import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.sql.SQLException;
import java.util.Properties;

public class IncomingServer extends Thread
{
    public int host_port = 0;
    public BufferedReader in = null;
    public PrintWriter out = null;
    public ServerSocket servers = null;
    public Socket fromclient = null;
    public DB assistant;
    public int UserId;
    public int UIDVALIDITY = 0;
    public String folder;
    public static final String[] CAPABILITY = new String[]{"capability",
"CAPABILITY"};
    public static final String[] LOGIN = new String[]{"login", "LOGIN"};
    public static final String[] SELECT = new String[]{"select", "SELECT"};
    public static final String[] SEARCH = new String[]{"search", "SEARCH"};
    public static final String[] FETCH = new String[]{"fetch", "FETCH"};
    public static final String[] NOOP = new String[]{"noop", "NOOP"};
    public static final String[] LOGOUT = new String[]{"logout", "LOGOUT"};
    public static final String[] LSUB = new String[]{"lsub", "LSUB"};
    public static final String[] LIST = new String[]{"list", "LIST"};
    public static final String[] CREATE = new String[]{"create", "CREATE"};
    public static final String[] SUBSCRIBE = new String[]{"subscribe",
"SUBSCRIBE"};
    public static final String[] COPY = new String[]{"copy", "COPY"};
    public static final String[] STATUS = new String[]{"status", "STATUS"};
    public static final String[] STORE = new String[]{"store", "STORE"};
    public static final String[] FLAGS = new String[]{"Answered", "Recent",
"Deleted", "Seen", "Draft", "Flagged"};
```

```
    /**
     * Конструктор IncomingServer
     * @param host_port - порт, на котором развернут сервер
     */
    public IncomingServer(int host_port)
    {
        this.host_port = host_port;
        //make new assistant for work with base
        try {
            assistant = new DB();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();  //To change body of catch statement use
File | Settings | File Templates.
        }
    }

    /**
     * Конструктор IncomingServer по-умолчанию
     */
    public IncomingServer() {
    }

    /**
     *Запуск сервера
     * Создает соединение, запускает процесс обмена сообщениями между
клиентом и сервером
     */
    public void run()
    {

        System.out.println("Server at " + host_port + " running");
        //creating server socket
        try {
            servers = new ServerSocket(host_port);
        } catch (IOException e) {
            System.err.println("Couldn't listen to port "+host_port);
            System.exit(-1);
        }
        // connecting client
        try {
            System.out.println("Waiting for a client...");
            fromclient = servers.accept();
            System.out.println("Client connected");
        } catch (IOException e) {
            System.err.println("Can't accept");
            System.exit(-1);
        }
        //Buffer for echo
        try {
            in  = new BufferedReader(new
                    InputStreamReader(fromclient.getInputStream()));
            out = new PrintWriter(fromclient.getOutputStream(),true);
        } catch (IOException e) {
            System.err.println("Couldn't make buffer");
            System.exit(-1);
        }

        //chating
        out.println("* OK, IMAP4 service is ready");

        int i = 0;
        while (i==0)
            try {
                WorkWithClient();
```

```
            } catch (SQLException e) {
                e.printStackTrace();  //To change body of catch statement use
File | Settings | File Templates.
            }

    }

    /**
     * Прием сообщения от клиента и передача его методу анализа сообщений
     * @throws SQLException - ошибки при анализе
     */
    public void WorkWithClient() throws SQLException {
        String client_message = null;
        try {
            client_message = in.readLine();
        } catch (IOException e) {
            System.err.println("Couldn't read client message");
            System.exit(-1);
        }
        if (client_message!=null)
        {
            System.out.println("Clent says: " + client_message);
            AnalyzeMessage(client_message);
        }
    }

    /**
     * закрытие соединения с клиентом
     */
    public void CloseServer()
    {
        //closing all thigs
        out.close();
        try {
            in.close();
        } catch (IOException e) {
            System.err.println("Couldn't close BufferedReader");
            System.exit(-1);
        }
        try {
            fromclient.close();
        } catch (IOException e) {
            System.err.println("Couldn't close ClientSocket");
            System.exit(-1);
        }
        try {
            servers.close();
        } catch (IOException e) {
            System.err.println("Couldn't close ServerSocket");
            System.exit(-1);
        }
    }

    /**
     * Анализ входящих сообщений
     * Ищем во входящей строке ключевое слово, если оно находится, запускаем
соответсвующий метод
     * @param client_message - сообщение от клиента
     * @throws SQLException - ошибки БД
     */
    public void AnalyzeMessage(String client_message) throws SQLException {
        String prefix = client_message.substring(0,client_message.indexOf("
"));
         //checking words
```

```
          if (client_message.contains(CAPABILITY[0]) ||
client_message.contains(CAPABILITY[1]) )
          {
              CapabilityCommand capability = new CapabilityCommand(prefix,
out);
              capability.DoCommand();
          }

          if (client_message.contains(LOGIN[0]) ||
client_message.contains(LOGIN[1]) )
          {
              LoginCommand login = new LoginCommand(prefix, client_message,
assistant, out);
              login.DoCommand();
              UserId = login.UserId;
          }

         if (client_message.contains(SELECT[0]) ||
client_message.contains(SELECT[1]) )
         {
             SelectCommand select = new SelectCommand(prefix, client_message,
assistant, out, UserId, UIDVALIDITY);
             select.DoCommand();
             folder = select.folder;
             UIDVALIDITY = select.UIDVALIDITY;
         }

         if (client_message.contains(SEARCH[0]) ||
client_message.contains(SEARCH[1]) )
         {
             SearchCommand search = new SearchCommand(prefix, client_message,
assistant, out, UserId);
             search.DoCommand();
         }

         if (client_message.contains(FETCH[0]) ||
client_message.contains(FETCH[1]) )
         {
             FetchCommand fetch = new FetchCommand(prefix, client_message,
assistant, out, folder, UserId);
             fetch.DoCommand();
         }

         if (client_message.contains(NOOP[0]) ||
client_message.contains(NOOP[1]) )
         {
             NoopCommand noop = new NoopCommand(prefix, out);
             noop.DoCommand();
         }

         if (client_message.contains(LOGOUT[0]) ||
client_message.contains(LOGOUT[1]) )
         {
             LogoutCommand logout = new LogoutCommand(prefix, out);
             logout.DoCommand();
         }

         if (client_message.contains(LSUB[0]) ||
client_message.contains(LSUB[1]) )
         {
             LsubCommand lsub = new LsubCommand(prefix, out);
             lsub.DoCommand();
         }
```

```
        if (client_message.contains(LIST[0]) ||
client_message.contains(LIST[1]) )
        {
            ListCommand list = new ListCommand(prefix, client_message, out,
assistant, UserId);
            list.DoCommand();
        }

        if (client_message.contains(CREATE[0]) ||
client_message.contains(CREATE[1]) )
        {
            CreateCommand create = new CreateCommand(prefix, client_message,
UserId, out, assistant);
            create.DoCommand();
        }

        if (client_message.contains(SUBSCRIBE[0]) ||
client_message.contains(SUBSCRIBE[1]) )
        {
            SubscribeCommand subscribe = new SubscribeCommand(prefix, out);
            subscribe.DoCommand();
        }

        if (client_message.contains(COPY[0]) ||
client_message.contains(COPY[1]) )
        {
            CopyCommand copy = new CopyCommand(prefix, client_message,
assistant, out);
            copy.DoCommand();
        }

        if (client_message.contains(STATUS[0]) ||
client_message.contains(STATUS[1]) )
        {
            StatusCommand status = new StatusCommand(prefix, out,
client_message, UserId, assistant);
            status.DoCommand();
        }


        if (client_message.contains(STORE[0]) ||
client_message.contains(STORE[1]) )
        {
            StoreCommand store = new StoreCommand(prefix, client_message,
assistant, out);
            store.DoCommand();
        }

     }

    /**
     * Извлечения первого параметра из сообщения клиента
     * @param request - сообщение клиента
     * @return первый параметр
     */
    public String GetFirstParamFromRequest(String request)
    {
        return request.substring(
(request.indexOf("\"")+1),request.indexOf("\"",request.indexOf("\"")+1));
    }
    /**
     * Извлечения второго параметра из сообщения клиента
     * @param request - сообщение клиента
     * @return второй параметр
```

```
     */
    public String GetSecondParamFromRequest(String request)
    {
        int indexSecondseparator  =
request.indexOf("\"",request.indexOf("\"")+1);
        return request.substring(indexSecondseparator + 3,request.length()-1)
;
    }

    /**
     * Открытие сообщения с диска
     * @param UID - UID сообщения
     * @return открытое сообщение
     */
    public MimeMessage OpenMessage (String UID)
    {
        //Connect *.eml file
        String emlFile = "C:\\IMAP4\\mail\\" + UID + ".eml";
        Properties props = System.getProperties();
        Session mailSession = Session.getDefaultInstance(props, null);
        InputStream source = null;
        try {
            source = new FileInputStream(emlFile);
        } catch (FileNotFoundException e) {
            System.err.println("Can't find this way ");
            e.printStackTrace();
        }
        MimeMessage message = null;
        try {
            message = new MimeMessage(mailSession, source);
        } catch (MessagingException e) {
            e.printStackTrace();
        }

        return message;
    }

    /**
     * Отправка сообщения клиенту и вывод его в консоль
     * @param repsonse - ответ для клиента
     * @param out - поток для общения с клиентом
     */
    public void SendAndPrint (String repsonse, PrintWriter out)
    {//just for typing less
        out.println (repsonse);
        System.out.println ("IncomingServer says: " + repsonse);
    }

 }
```

Класс DB

```
package DBHandler;

import java.sql.*;
import java.util.Properties;

public class DB
{
    public String DB_URL =
"jdbc:firebirdsql://localhost:3050/C:/IMAP4/DB.FDB";
    public String DB_DEFAULT_USER = "SYSDBA";
    public String DB_DEFAULT_PASSWORD = "masterkey";
```

```
    public String DB_DEFAULT_ENCODING = "win1251";
    public Connection conn = null;
    Properties props = null;

    /**
     * Конструктор DB
     * заполняем параметры, вызываем подключение
     * @throws ClassNotFoundException - ошибка jdbc
     */
    public DB() throws ClassNotFoundException
    {
        Class.forName("org.firebirdsql.jdbc.FBDriver");
        props = new Properties();
        props.setProperty("user", DB_DEFAULT_USER);
        props.setProperty("password", DB_DEFAULT_PASSWORD);
        props.setProperty("encoding", DB_DEFAULT_ENCODING);
        DBConnect();
    }

    /**
     * Подключение к БД
     */
    public void DBConnect()
    {
        try {
            conn = DriverManager.getConnection(DB_URL, props);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * Отключение от БД
     */
    public void BDDisconnect()
    {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * Поиск пользователя
     * @param login
     * @param password
     * @return resultSet
     */
    public ResultSet FindUser (String login, String password)
    {
        String Query = "SELECT USER_ID AS \"USER_ID\"" +
                " FROM USERS" +
                " WHERE LOGIN  = '" +  login + "' AND PASSWORD = '" +
password + "'";
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }
```

```
    /**
     * Считаем количество писем в конкретной папке
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return resultSet
     */
    public ResultSet CountExistMessages(String user_id, String folder)
    {
        String Query = "SELECT COUNT(1) AS \"COUNT\" FROM MESSAGES WHERE
USER_ID = '" + user_id + "'" +
                " AND FOLDER = '" + folder + "'";

        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Считаем недавние сообщения в папке
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return resultSet
     */
    public ResultSet CountRecentMessages (String user_id, String folder)
    {
        String Query = "SELECT COUNT(1) AS \"COUNT\" FROM MESSAGES WHERE
USER_ID = '" + user_id + "'" +
            " AND FLAG_RECENT = 1" +  " AND FOLDER = '" + folder + "'";

        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Считаем непрочитанные сообщения в папке
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return resultSet
     */
    public ResultSet CountUnSeenMessages (String user_id, String folder)
    {
        String Query = "SELECT COUNT(1) AS \"COUNT\" FROM MESSAGES WHERE
USER_ID = '" + user_id + "'" +
                " AND FLAG_SEEN = 0" + " AND FOLDER = '" + folder + "'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
```

```
        }
    }

    /**
     * Получаем ID первого непрочитанного сообщения
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return resultSet
     */
    public ResultSet FirstUnseen (String user_id, String folder)
    {
        String Query = "SELECT MESSAGE_ID AS \"MESSAGE_ID\" FROM MESSAGES
WHERE USER_ID = '" + user_id + "'" +
                " AND FLAG_SEEN = 0" + " AND FOLDER = '" + folder + "'" +
                " ORDER BY MESSAGE_ID ASCENDING";
        System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Получение флагов сообщения
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @param message_id - ID сообщения
     * @return resultSet
     */
    public ResultSet GetMessageFlags (String user_id, String folder, String
message_id)
    {
        String Query = "SELECT FLAG_SEEN AS \"SEEN\"," +
                " FLAG_ANSWERED AS \"ANSWERED\"," +
                " FLAG_DELETED AS \"DELETED\"," +
                " FLAG_DRAFT AS \"DRAFT\"," +
                " FLAG_RECENT AS \"RECENT\"" +
                " FROM MESSAGES WHERE USER_ID = '" + user_id + "'" +
                " AND MESSAGE_ID ='" + message_id + "'" +
                " AND FOLDER = '" + folder + "'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Считаем количество сообщений с флагом в определенной папке
     * @param user_id - ID пользователя
     * @param folder - - имя папки
     * @param flag - флаг
     * @return resultSet
     */
    public ResultSet CountFlageddMessages (String user_id, String folder,
String flag)
```

```
    {
        String Query = "SELECT MESSAGE_ID AS \"MESSAGE_ID\" FROM MESSAGES
WHERE USER_ID = '" + user_id + "'" +
                " AND FLAG_" + flag +" = 1" + " AND FOLDER = '" + folder +
"'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Получение вложенных папок
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return resultSet
     */
    public ResultSet GetDeepFolser (String user_id, String folder)
    {
        String Query = "SELECT FOLDERS.FOLDER_NAME AS \"FOLDER_NAME\" FROM
FOLDERS" +
                " LEFT OUTER JOIN USERS" +
                " ON FOLDERS.USER_ID  = USERS.USER_ID" +
                " WHERE FOLDERS.UPPER_FOLDER_NAME = '" + folder + "'" +
                " AND FOLDERS.USER_ID = '" + user_id + "'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Поиск папки
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return resultSet
     */
    public ResultSet SearchFolder (String user_id, String folder)
    {
        String Query = "SELECT FOLDERS.FOLDERS_ID AS \"FOLDERS_ID\" FROM
FOLDERS" +
                " LEFT OUTER JOIN USERS" +
                " ON FOLDERS.USER_ID  = USERS.USER_ID" +
                " WHERE FOLDERS.FOLDER_NAME = '" + folder + "'" +
                " AND FOLDERS.USER_ID = '" + user_id + "'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
```

```
    }

    /**
     * Генерируем новое ID для сообщения
     * @return resultSet
     */
    public ResultSet GetMessageUid ()
    {
        String Query = "SELECT GEN_ID(GEN_MESSAGES_ID, 0) FROM RDB$DATABASE";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            ResultSet resultSet = statement.executeQuery();
            return resultSet;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Добавляем новый флаг для сообщения
     * @param message_id - ID сообщения
     * @param flag - имя флага
     * @return 1 - если успешно
     *         0 - если не успешно
     */
    public int AddFlagToMessage (String message_id, String flag)
    {
        String Query = "UPDATE MESSAGES SET FLAG_" + flag + " = '1'" +
                " WHERE MESSAGE_ID = '" + message_id + "'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            //return statement.execute();
            return  statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * Удаление флага из сообщения
     * @param message_id - ID сообщений
     * @param flag - имя флага
     * @return 1 - если успешно
     *         0 - если не успешно
     */
    public int RemoveFlagFromMessage (String message_id, String flag)
    {
        String Query = "UPDATE MESSAGES SET FLAG_" + flag + " = '0'" +
                " WHERE MESSAGE_ID = '" + message_id + "'";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            //return statement.execute();
            return  statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }
```

```
    /**
     * Добавление папки
     * @param user_id - ID пользователя
     * @param folder - имя папки
     * @return 1 - если успешно
     *         0 - если не успешно
     */
    public int AddFolderToUser (String user_id, String folder)
    {
        String Query = "INSERT INTO FOLDERS (FOLDER_NAME, USER_ID,
FOLDERS_ID)" +
                " VALUES ('" + folder + "', '" + user_id + "', (SELECT
GEN_ID(GEN_FOLDERS_ID, 1) FROM RDB$DATABASE))";
        //System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            return  statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }

    /**
     * Копирование сообщения в папку
     * @param message_id - ID сообщения
     * @param folder - имя папки назначения
     * @return 1 - если успешно
     *         0 - если не успешно
     */
    public int CopyMessage (String message_id, String folder)
    {
        String Query = "UPDATE MESSAGES SET FOLDER = '" + folder + "'" +
                " WHERE MESSAGE_ID = '" + message_id + "'";
        System.out.println(Query);
        try {
            PreparedStatement statement = conn.prepareStatement(Query);
            //return statement.execute();
            return  statement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
            return 0;
        }
    }
}
```

Класс SubscribeCommand

```
package Commands;

import Server.IncomingServer;

import java.io.PrintWriter;


public class SubscribeCommand extends IncomingServer {
    public String prefix;
    public PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;

    public SubscribeCommand (int host_port) {//default constructor
        super(host_port);
```

```
    }

    public SubscribeCommand (String prefix, PrintWriter out) {
        this.prefix = prefix;
        this.out = out;
    }

    public void DoCommand () {
        markedResponse = MakeMarked();
        SendAndPrint(markedResponse, out);
    }

    public String MakeMarked () {
        String response = prefix + " BAD SUBSCRIBE illegal command";
        return response;
    }
}
```

Класс StoreCommand

```
package Commands;

import DBHandler.DB;
import Server.IncomingServer;

import java.io.PrintWriter;
import java.util.Vector;

public class StoreCommand extends IncomingServer {

    public String prefix;
    public String request;
    public DB assistant;
    public String unmarkedResponse;
    public String markedResponse;
    public Boolean plus;
    public PrintWriter out;
    /**
     * Конструктор StoreCommand
     * @param host_port - порт на котором работает сервер
     */
    public StoreCommand (int host_port) {//default constructor
        super(host_port);
    }
    /**
     * Конструктор StoreCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     */
    public StoreCommand (String prefix, String request, DB assistant,
PrintWriter out) {//no-args constructor
        this.prefix = prefix;
        this.request = request;
        this.assistant = assistant;
        this.out = out;
    }
    /**
     * Выполнение комманды
     * Сохранение письма в БД
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
```

```
    public void DoCommand () {
        GetPluse ();
        String UID = GetUID();
        Vector <String> flags = GetFlags();


        unmarkedResponse = MakeUnmarked(UID, flags);
        markedResponse = MakeMarked(UID, flags);

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);


    }

    /**
     * Проверка наличия флага "+" в запросе
     */
    public void GetPluse () {
        if (request.contains("+"))
        {
            plus = true;
        }
        else
        {
            plus = false;
        }
    }

    /**
     * Получение UID письма
     * @return UID
     */
    public String GetUID () {
        String UID ="";
        int startSymbol = request.indexOf('E') + 2;
        int endSymbol;

        if (plus)
        {
            endSymbol = request.indexOf('+') -2;
        }
        else
        {
            endSymbol = request.indexOf('-') -2;
        }

        for (int i = startSymbol; i<= endSymbol; i++)
        {
            UID += request.charAt(i);
        }

        return UID;
    }

    /**
     * Получения флагов письма
     * @return флаги
     */
    public Vector<String> GetFlags () {
        Vector <String> flags = new Vector<String>();

        for (int i = 0; i < FLAGS.length; i++)
        {
```

```
            if (request.contains(FLAGS[i]))
            {
                flags.add(FLAGS[i]);
            }
        }

        return flags;
    }

    /**
     * Добавляем флаги сообщения в БД
     * @param UID - UID письма
     * @param flag - флаги
     * @return - 1 - если сохранение успешно
     *           2 - если произошла ошибка
     */
    public int ChangeFlag (String UID, String flag){
        String uppercaseFlag = flag.toUpperCase();
        int result;
        if (plus)
        {
            result = assistant.AddFlagToMessage(UID,uppercaseFlag);
        }
        else
        {
            result = assistant.RemoveFlagFromMessage(UID, uppercaseFlag);
        }

        return result;
    }
    /**
     * Собираем строку ответа без префикса
     * @param UID - UID сообщения для копирования
     * @param flags - флайги сообщения
     * @return строка с ответом
     */
    public String MakeUnmarked (String UID, Vector <String> flags) {
        String response = ("* STORE " + UID + " FLAGS (");
        for (int i = 0; i< flags.size(); i++)
        {
            response += "\\" + flags.get(i) + " ";
        }
        response = response.substring(0,response.length()-1);
        response += ")";

        return response;
    }
    /**
     * Собираем строку ответа с префиксом
     * @param UID - UID сообщения для копирования
     * @param flags - флайги сообщения
     * @return строка с ответом
     */
    public String MakeMarked (String UID, Vector <String> flags) {
        String response = "";

        for (int i = 0; i< flags.size(); i++)
        {
            int result = ChangeFlag(UID, flags.get(i));
            if (result == 1)
            {
                response = prefix + " OK STORE completed";
            }
            else
```

```
            {
                response = prefix + " NO STORE can't complete";
            }
        }
        return response;
    }
}
```

Класс StatusCommand

```
package Commands;

import DBHandler.DB;
import Server.IncomingServer;

import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;

public class StatusCommand extends IncomingServer {
    public String prefix;
    public PrintWriter out;
    public String request;
    public int UserId;
    public DB assistant;

    public String folder;
    public String userId;
    public String unmarkedResponse;
    public String markedResponse;

    /**
     * Конструктор StatusCommand
     * @param host_port
     */
    public StatusCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор StatusCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     * @param UserId - ID пользователя
     */
    public StatusCommand (String prefix, PrintWriter out, String request, int
UserId, DB assistant) {//no-args constructor
        this.prefix = prefix;
        this.out = out;
        this.request = request;
        this.UserId = UserId;
        this.assistant = assistant;
    }

    /**
     * Выполнение комманды
     * Работа со статусами сообщений с БД
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        folder = GetFirstParamFromRequest(request);
```

```
        userId = Integer.toString(UserId);

        markedResponse = MakeMarked();
        unmarkedResponse = MakeUnmarked();

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);
    }
    /**
     * Собираем строку ответа без префикса
     * @return строка с ответом
     */
    public String MakeUnmarked () {
        String response = prefix + " OK STATUS completed";

        return response;
    }
    /**
     * Собираем строку ответа с префиксом
     * Обращаемся к БД что бы получить статусы писем в папке
     * @return строка с ответом
     */
    public String MakeMarked () {
        String response = "* STATUS " + folder + " (";

        if (request.contains("UIDNEXT"))
        {
            ResultSet resultSet = assistant.GetMessageUid();
            try {
                if (resultSet.next())
                {
                    String uid = resultSet.getString(1);
                    response += "UIDNEXT " + uid;
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if (request.contains("MESSAGES"))
        {
            ResultSet resultSet = assistant.CountExistMessages(userId,
folder);
            try {
                if (resultSet.next())
                {
                    String exist = resultSet.getString("COUNT");
                    response += " MESSAGES " + exist;
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if (request.contains("UNSEEN"))
        {
            ResultSet resultSet = assistant.CountUnSeenMessages(userId,
folder);
            try {
                if (resultSet.next())
                {
                    String unseen = resultSet.getString("COUNT");
                    response += " UNSEEN " + unseen;
                }
```

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

        if (request.contains("RECENT"))
        {
            ResultSet resultSet = assistant.CountRecentMessages(userId,
folder);
            try {
                if (resultSet.next())
                {
                    String recent = resultSet.getString("COUNT");
                    response += " RECENT " + recent;
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        response += ")";

        return response;
    }

}
```

Класс SelectCommand

```
public class SelectCommand extends IncomingServer {
    public String prefix;
    public String request;
    public DB assistant;
    public String unmarkedResponse;
    public String markedResponse;
    public PrintWriter out;

    public String folder;
    public int existMessages;
    public int recentMessages;
    public int unSeenMessages;
    public int UserId;
    public int UIDVALIDITY;

    /**
     * Конструктор SelectCommand
     * @param host_port - порт на котором работает сервер
     */
    public SelectCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор SelectCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     * @param UserId - ID пользователя
     * @param UIDVALIDITY - UID запроса
     */
    public SelectCommand (String prefix, String request, DB assistant,
PrintWriter out, int UserId, int UIDVALIDITY) {
```

```
        this.prefix = prefix;
        this.request = request;
        this.assistant = assistant;
        this.out = out;
        this.UserId = UserId;
        this.UIDVALIDITY = UIDVALIDITY;
    }

    /**
     * Выполнение комманды
     * Увеличиваем UIDVALIDITY, работаем с БД
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        folder = GetFirstParamFromRequest(request);
        UIDVALIDITY++; //geting new UIDVALIDITY

        markedResponse = MakeMarked();
        unmarkedResponse = MakeUnmarked();

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);

    }
    /**
     * Собираем строку ответа с префиксом
     * Получаем статистику по сообщением из БД
     * @return строка с ответом
     */
    public String MakeMarked () {
        String response = "";
        String userId = Integer.toString(UserId);
        ResultSet resultSet;
        resultSet = assistant.CountExistMessages(userId,folder);
        try {
            if (resultSet.next())
                existMessages =
Integer.valueOf(resultSet.getString("COUNT"));
            else
            {
                System.err.println("DB error");
                response = prefix + " NO SELECT failed";
                return response;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        resultSet = assistant.CountRecentMessages(userId,folder);
        try {
            if (resultSet.next())
                recentMessages =
Integer.valueOf(resultSet.getString("COUNT"));
            else
            {
                System.err.println("DB error");
                response = prefix + " NO SELECT failed";
                return response;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        resultSet = assistant.CountUnSeenMessages(userId,folder);
```

```
        try {
            if (resultSet.next())
                unSeenMessages =
Integer.valueOf(resultSet.getString("COUNT"));
            else
            {
                System.err.println("DB error");
                response = prefix + " NO SELECT failed";
                return response;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        response = prefix + " OK [READ-WRITE] SELECT Completed";

        return response;
    }
    /**
     * Собираем строку ответа без префикса
     * @return строка с ответом
     */
    public String MakeUnmarked () {

        String firstUnseen = FirstUnseen();

        String response = "* " + existMessages + " EXISTS\n" +
                "* " + recentMessages +" RECENT\n" +
                "* OK [UNSEEN " + unSeenMessages + "] " + firstUnseen + "\n"+
                "* OK [UIDVALIDITY " + UIDVALIDITY + "] UIDs valid\n" +
                "* FLAGS (\\Answered \\Recent \\Deleted \\Seen \\Draft
\\Flagged)\n" +
                "* OK [PERMANENTFLAGS (\\Deleted \\Seen \\*)] Limited";

        return  response;
    }

    /**
     * Получаем строку-ответ с первым непрочитанным сообщением
     * @return
     */
    public String FirstUnseen ()
    {
        String response = "";
        ResultSet resultSet = assistant.FirstUnseen(Integer.toString(UserId),
folder);

        try {
            if (resultSet.next())
            {
                response = "Message " + resultSet.getString("MESSAGE_ID") + "
is first unseen";
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return response;
    }
}
```

Класс SearchCommand

```
package Commands;
```

```java
import DBHandler.DB;
import Server.IncomingServer;

import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;

public class SearchCommand extends IncomingServer {

    public DB assistant;
    public int UserId;
    public String request;
    public String prefix;
    public PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;
    /**
     * Конструктор SearchCommand
     * @param host_port - порт на котором работает сервер
     */
    public SearchCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор SearchCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     * @param UserId - ID пользователя
     */
    public SearchCommand (String prefix, String request, DB assistant,
PrintWriter out, int UserId) {//no-args constructor
        this.prefix = prefix;
        this.request = request;
        this.assistant = assistant;
        this.out = out;
        this.UserId = UserId;
    }
    /**
     * Выполнение комманды
     * Отправляем запрос БД для поиска
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        unmarkedResponse = MakeUnmarked();
        markedResponse = MakeMarked();

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);
    }

    /**
     * Собираем строку ответа без префикса
     * результат получаем от БД
     * @return строка с ответом
     */
    public String MakeUnmarked () {
        String response = "* SEARCH";
        String userId = Integer.toString(UserId);
        ResultSet resultSet = null;
```

```
        if (request.indexOf("DELETED")!=-1)
        {
            resultSet = assistant.CountFlageddMessages(userId, folder,
"DELETED");
        }

        if (request.indexOf("SEEN")!=-1)
        {
            resultSet = assistant.CountFlageddMessages(userId, folder,
"SEEN");
        }

        try {
            while (resultSet.next())
            {
                response +=  " " + resultSet.getString("MESSAGE_ID");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

      return response;
    }
    /**
     * Собираем строку ответа с префиксом
     * @return строка с ответом
     */
    public String MakeMarked () {
        String response = prefix + " OK SEARCH completed";

       return response;
    }

}
```

Класс NoopCommand

```
package Commands;

import Server.IncomingServer;

import java.io.PrintWriter;

public class NoopCommand extends IncomingServer {
    public String prefix;
    public PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;
    /**
     * Конструктор NoopCommand
     * @param host_port - порт на котором работает сервер
     */
    public NoopCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор NoopCommand
     * @param prefix - префикс команды
     * @param out - поток для ответа
     */
    public NoopCommand (String prefix, PrintWriter out) {
```

```
        this.prefix = prefix;
        this.out = out;
    }

    /**
     * Выполнение команды
     * Отправляем ответ NOOP
     */
    public void DoCommand () {
        markedResponse = MakeMarked();

        SendAndPrint(markedResponse, out);
    }
    /**
     * Собираем строку ответа с префиксом
     * @return строка с ответом
     */
    public String MakeMarked () {
        String response = prefix + " OK NOOP completed";
        return response;
    }

}
```

Класс LSubCommand

```
package Commands;

import Server.IncomingServer;

import java.io.PrintWriter;


public class LsubCommand extends IncomingServer {
    public String prefix;
    public PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;
    /**
     * Конструктор LsubCommand
     * @param host_port - порт на котором работает сервер
     */
    public LsubCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор LsubCommand
     * @param prefix - префикс команды
     * @param out - поток для ответа
     */
    public LsubCommand (String prefix, PrintWriter out) {
        this.prefix = prefix;
        this.out = out;
    }
    /**
     * Выполнение комманды
     * отправляем ответ о том, что наш сервер не поддерживает данную комманду
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        markedResponse = MakeMarked();
        SendAndPrint(markedResponse, out);
```

```
    }
    /**
     * Собираем строку ответа с префиксом
     * @return строка с ответом
     */
    public String MakeMarked () {
        String response = prefix + " BAD LSUB illegal command";
        return response;
    }
}
```

Класс LogoutCommand

```
package Commands;

import Server.IncomingServer;

import java.io.PrintWriter;

public class LogoutCommand extends IncomingServer {
    public String prefix;
    public PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;

    /**
     * Конструктор LogoutCommand
     * @param host_port - порт на котором работает сервер
     */
    public LogoutCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор LogoutCommand
     * @param prefix - префикс команды
     * @param out - поток для ответа
     */
    public LogoutCommand (String prefix, PrintWriter out) {
        this.prefix = prefix;
        this.out = out;
    }

    /**
     * Выполнение команды
     * Выход пользователя.
     */
    public void DoCommand () {
        markedResponse = MakeMarked();
        unmarkedResponse = MakeUnmarked();

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);
    }

    /**
     * Собираем строку ответа без префикса
     * @return строка с ответом
     */
    public String MakeUnmarked () {
        String response = "* BYE IMAP4rev1 server terminating connection";

        return response;
```

```
    }

    /**
     * Собираем строку ответа с префиксом
     * @return строка с ответом
     */
    public String MakeMarked () {
        String response = prefix + " LOGOUT completed";

        return response;
    }

}
```

Класс LoginCommand

```
package Commands;


import DBHandler.DB;
import Server.IncomingServer;

import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;

public class LoginCommand extends IncomingServer {
    public DB assistant;
    public String prefix;
    public String request;
    public PrintWriter out;
    public int UserId;
    public String markedResponse;

    /**
     * Конструктор LoginCommand
     * @param host_port - порт на котором работает сервер
     */
    public LoginCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор LoginCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     */
    public LoginCommand (String prefix, String request, DB assistant,
PrintWriter out) {//no-args constructor
        this.prefix = prefix;
        this.request = request;
        this.assistant = assistant;
        this.out = out;
    }

    /**
     * Выполнение комманды
     * Проверка пользователя
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        Boolean existUser = FindUser ();
        markedResponse = MakeMarked(existUser);
```

```
        SendAndPrint(markedResponse, out);
    }

    /**
     * Собираем строку ответа без префикса
     * @param existUser - существует ли пользователь
     * @return строка с ответом
     */
    public String MakeMarked (Boolean existUser) {
        String response = "";
        if (existUser)
        {
           response = prefix + " OK LOGIN completed";
        }
        else
        {
            response = prefix + " NO LOGIN failed";
        }

        return response;
    }

    /**
     * Ищем указанного в запросе пользователя
     * @return найден ли пользователь
     */
    public Boolean FindUser ()
    {
        Boolean user = null;
        String login = GetFirstParamFromRequest(request);
        String password = GetSecondParamFromRequest(request);

        ResultSet resultSet = assistant.FindUser(login,password);

        try {
            if (resultSet.next())
            {
                UserId = Integer.valueOf(resultSet.getString("USER_ID"));
                user = true;
            }
            else
            {
                user = false;
            }
        } catch (SQLException e) {
            e.printStackTrace();  //To change body of catch statement use
File | Settings | File Templates.
        }

        return user;
    }
}
```

Класс ListCommand

```
package Commands;

import DBHandler.DB;
import Server.IncomingServer;

import java.io.PrintWriter;
import java.sql.ResultSet;
```

```java
import java.sql.SQLException;

public class ListCommand extends IncomingServer {
    public String prefix;
    public String request;
    public DB assistant;
    public PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;
    public int UserId;
    String firstParam;
    String secondParam;

    /**
     * Конструктор ListCommand
     * @param host_port
     */
    public ListCommand (int host_port) {//default constructor
        super(host_port);
    }
    //we doing nothing, but can update something

    /**
     * Конструктор ListCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param out - поток для ответа
     * @param assistant - объект для работы с бд
     * @param UserId - ID пользователя
     */
    public ListCommand (String prefix, String request, PrintWriter out, DB
assistant, int UserId) {//no-args constructor
        this.prefix = prefix;
        this.out = out;
        this.request = request;
        this.assistant = assistant;
        this.UserId = UserId;
    }
    /**
     * Выполнение комманды
     * Вначале извлекаем из запроса необходимые поля и далее обращаемся к дб
для получения списка сообщений.
     * Полученный результат отправляем клиенту
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {

        firstParam =  GetFirstParamFromRequest(request);
        secondParam = GetSecondParamFromRequest(request);

        Boolean exists = FolderExist();

        markedResponse = MakeMarked(exists);
        if (exists)
        {
            unmarkedResponse = MakeUnmarked();
            SendAndPrint(unmarkedResponse, out);
        }

        SendAndPrint(markedResponse, out);
    }

    /**
     * Проверяем, существует ли указанная папка
```

```
     * @return - существует или нет
     */
    public Boolean FolderExist ()
    {
        Boolean exists = null;
        ResultSet resultSet =
assistant.SearchFolder(Integer.toString(UserId),secondParam);
        try {
            if (resultSet.next())
            {
                exists = true;
            }
            else
            {
                exists = false;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return  exists;
    }

    /**
     * Собираем строку ответа без префикса
     * @return
     */
    public String MakeUnmarked () {
        String response = "";
        ResultSet resultSet = null;
            if (firstParam.equals(""))
            {
                resultSet =
assistant.GetDeepFolser(Integer.toString(UserId),secondParam);
                try {
                    if (resultSet.next())
                    {
                        while (resultSet.next())
                        {
                            //NEED TO COMPLETE!!
                        }
                    }
                    else
                    {
                        response ="* LIST (\\NoInferiors) \"|\" " +
secondParam;
                    }

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

        return response;
    }

    /**
     * Собираем строку ответа с префиксом
     * @param exists - существует ли указанная папка
     * @return - строка с ответом
     */
    public String MakeMarked (Boolean exists) {
        String response = "";

        if (exists)
```

```
        {
            response = prefix + " OK LIST completed";
        }
        else
        {
            response = prefix + " BAD LIST can't find this mailbox";
        }

        return response;
    }

}
```

Класс FetchCommand

```
package Commands;

import DBHandler.DB;
import Server.IncomingServer;

import javax.mail.MessagingException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.io.*;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Vector;


public class FetchCommand extends IncomingServer {

    public String prefix;
    public String request;
    public DB assistant;
    PrintWriter out;
    public String folder;
    public int UserId;

    /**
     * Конструктор FetchCommand
     * @param host_port - порт на котором работает сервер
     */
    public FetchCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор FetchCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     * @param folder - рабочая папка
     * @param UserId - ID пользователя
     */
    public FetchCommand (String prefix, String request, DB assistant,
PrintWriter out,  String folder, int UserId) {//no-args constructor
        this.prefix = prefix;
        this.request = request;
        this.assistant = assistant;
        this.out = out;
        this.folder = folder;
        this.UserId = UserId;
```

```
    }

    /**
     * Выполнение команды
     * Разные клиенты присылают разного вида запросы. Логика обработки
запросов описана в комментариях
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand()  {
        String startUID = "";
        String endUID = ""; //see next comment
        Boolean sequence = false; // if we need to fetch something like 1:5
or 1:*
        Boolean allUID = false;// if we need to fetch :*
        Boolean uidIsVector = false;  // if we need to fetch 2,3,4

        Vector<String> vecUID = new Vector <String>();

        int separator = request.indexOf(':');
        if (separator > 0) //we have sequence
        {
            sequence = true;
            separator --;
            while (request.charAt(separator) != ' ')
            {
                startUID = request.charAt(separator) + startUID;
                separator --;
            }

            separator = request.indexOf(':')+ 1;

            if (request.charAt(separator) == '*')//if all UIDs
            {
                allUID = true;
            }
            else
            {
                while (request.charAt(separator) != ' ')
                {
                    endUID = request.charAt(separator) + endUID;
                    separator ++;
                }
            }
        }
        else //we need only 1 or a few
        {
            //if 1
            separator = request.indexOf (',');
            if (separator <= 0 ) //if only one
            {
                separator = request.indexOf ('H') + 2; // proplems with low
case

                while (request.charAt(separator) != ' ')
                {
                    startUID += request.charAt(separator) ;
                    separator ++;
                }
            }
            else  //if a few
            {
                uidIsVector = true;
                String tempUID = "";
                separator --;
```

```
                while (request.charAt(separator) != ' ')
                {
                    tempUID = request.charAt(separator) + tempUID;
                    separator --;
                }
                vecUID.add(tempUID);
                separator = request.indexOf(',')+1;
                tempUID = "";
                while (request.charAt(separator) != ' ')
                {
                    if (request.charAt(separator) !=',')
                    {
                        tempUID  = request.charAt(separator) + tempUID;
                        separator ++;
                    }
                    else
                    {
                        vecUID.add(tempUID);
                        separator ++;
                        tempUID = "";
                    }
                }
                vecUID.add(tempUID);
            }
        }

        if (sequence)
        {
            int startuid = Integer.valueOf(startUID);
            String userId = Integer.toString(UserId);
            ResultSet resultSet =
assistant.CountExistMessages(userId,folder);
            int count = 0;
            try {
                if (resultSet.next())
                {
                    count = Integer.valueOf(resultSet.getString("COUNT"));
                }
                else
                {
                    System.out.println("ERRROR");
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }


            if (allUID)
            {
                if (count <= 0)
                {
                    SendAndPrint(prefix + " No FETCH no one message", out);
                    return;
                }

                for ( int i = startuid ; i<= count; i++)
                {
                    try {

GenAndSendFetchResponse(request,Integer.toString(i),sequence);
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
```

```
            }
            else
            {
                int enduid = Integer.valueOf(endUID);
                int finishUID;
                if (enduid <= count)
                {
                    finishUID = enduid;
                }
                else
                {
                    SendAndPrint(prefix + " No FETCH in folder only " + count
+ " messages", out);
                    return;
                }
                for ( int i = startuid ; i<= finishUID; i++)
                {
                    try {

GenAndSendFetchResponse(request,Integer.toString(i),sequence);
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }

            }
        }
        else
        {
            if (uidIsVector) //if 2,3,4
            {
                for (int i = 0; i< vecUID.size(); i++)
                {
                    try {

GenAndSendFetchResponse(request,vecUID.get(i),sequence);
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
            }
            else // if 1
            {
                try {
                    GenAndSendFetchResponse(request,startUID,sequence);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }

       SendAndPrint(prefix + " OK FETCH completed", out);
    }

    /**
     * Генерирование ответа на Fetch запрос
     * @param request - строка запроса
     * @param UID
     * @param sequence - если сообщений несколько
     * @throws SQLException - при ошибке работы с БД
     */
    public void GenAndSendFetchResponse (String request, String UID, Boolean
sequence) throws SQLException {
```

```
        Boolean  needbracket = true;
        int separator = 10; //this number is roughly
        //open message
        MimeMessage message = OpenMessage (UID);

        String response = "* " + UID + " FETCH (";
        if (sequence)
        {
            response += "UID " + UID + " ";
        }

        //If UID command in Fetch
        if (request.indexOf("UID", separator)!=-1)
            response += "UID " + UID + " ";

        //If RFC822.SIZE command in Fetch
        if (request.indexOf("RFC822.SIZE", separator)!=-1)
            try {
                response += "RFC822.SIZE " + message.getSize() + " ";
            } catch (MessagingException e) {
                e.printStackTrace();
            }

        //If INTERNALDATE command in Fetch
        //INTERNALDATE = senddate but in real = recivedate
        if (request.indexOf("INTERNALDATE", separator)!=-1)
            try {
                response += "INTERNALDATE \"" + message.getSentDate() + "\"
";
            } catch (MessagingException e) {
                e.printStackTrace();
            }

        //If FLAGS command in Fetch
        if (request.contains("FLAGS"))
        {
            ResultSet resultSet =
assistant.GetMessageFlags(Integer.toString(UserId), folder, UID);
            String space = "";
            if (resultSet.next())
            {
                response += "FLAGS (";
                if ((Integer.parseInt(resultSet.getString("SEEN"))) == 1)
                {
                    response += "\\Seen";
                    space = " ";
                }
                if ((Integer.parseInt(resultSet.getString("ANSWERED"))) == 1)
                {
                    response += space + "\\Answered";
                    space = " ";
                }
                if ((Integer.parseInt(resultSet.getString("DELETED"))) == 1)
                {
                    response += space + "\\Deleted";
                    space = " ";
                }
                if ((Integer.parseInt(resultSet.getString("DRAFT"))) == 1)
                {
                    response += space + "\\Draft";
                    space = " ";
                }
                if ((Integer.parseInt(resultSet.getString("Recent"))) == 1)
                {
```

```
                    response += space + "\\Recent";
                    space = " ";
                }
                response += ") ";
            }
        }

        //If ENVELOPE command in Fetch
        if (request.indexOf("ENVELOPE", separator)!=-1)
        {
           // EnvelopeCommand env = new EnvelopeCommand(message);
            String envelope = null;
            try {
                envelope = getEnvelop(message);
            } catch (MessagingException e) {
                e.printStackTrace();
            }
            response += envelope + " ";
        }

        //If ENVELOPE command in Fetch
        if (request.indexOf("BODYSTRUCTURE", separator)!=-1)
        {
            //BodyStructureCommand bod = new BodyStructureCommand(message);
            String bodyStructure = null;
            try {
                bodyStructure = GetBodyStructure(message);
                bodyStructure.toUpperCase();
            } catch (MessagingException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            response += bodyStructure;
        }

        if (request.indexOf("BODY.PEEK[HEADER.FIELDS", separator)!=-1 ||
request.indexOf("BODY.PEEK[]", separator)!=-1)
        {
            needbracket = false;
            String emlFile = "D:\\mail\\" + UID + ".eml";
            String strLine = null;
            String line = "";
            try{
                // Open the file that is the first
                // command line parameter
                FileInputStream fstream = new FileInputStream(emlFile);
                // Get the object of DataInputStream
                DataInputStream in = new DataInputStream(fstream);
                BufferedReader br = new BufferedReader(new
InputStreamReader(in));
                //Read File Line By Line

                while ((strLine = br.readLine()) != null)   {
                    line += strLine + " ";
                    // Print the content on the console
                }
                //Close the input stream
                in.close();
            }catch (Exception e){//Catch exception if any
                System.err.println("Error: " + e.getMessage());
            }

            response += "BODY.PEEK[] {" + line.length() + "} \r\n";
```

```
            try {
                InternetAddress[] addrFrom = (InternetAddress[])
message.getFrom();
                if (addrFrom !=null)
                {
                    response += "From: ";
                    String space = "";
                    for (int i = 0; i<  addrFrom.length; i++)
                    {
                        response +=  space + addrFrom[i].getPersonal() + " <"
+ addrFrom[i].getAddress() + ">";
                        space = ", ";
                    }
                    response += "\r\n";
                }

                InternetAddress[] adrrTo = (InternetAddress[])
message.getRecipients(MimeMessage.RecipientType.TO);
                if (adrrTo!=null)
                {
                    response += "To: ";
                    String space = "";
                    for (int i = 0; i<  adrrTo.length; i++)
                    {
                        //ПОТОМ ИСПРАВИТЬ!
                        //response +=  space + adrrTo[i].getPersonal() + " "
+ adrrTo[i].getAddress();
                        response +=  adrrTo[i].getAddress();
                        space = ", ";
                    }
                    response += "\r\n";
                }

                InternetAddress[] adrrCc = (InternetAddress[])
message.getRecipients(MimeMessage.RecipientType.CC);
                if (adrrCc!=null)
                {
                    response += "Cc: ";
                    String space = "";
                    for (int i = 0; i<  adrrCc.length; i++)
                    {
                        response +=  space + adrrCc[i].getPersonal() + " " +
adrrCc[i].getAddress();
                        space = ", ";
                    }
                    response += "\r\n";
                }

                InternetAddress[] adrrBcc = (InternetAddress[])
message.getRecipients(MimeMessage.RecipientType.BCC);
                if (adrrBcc!=null)
                {
                    response += "Bcc: ";
                    String space = "";
                    for (int i = 0; i<  adrrBcc.length; i++)
                    {
                        response +=  space + adrrBcc[i].getPersonal() + " " +
adrrBcc[i].getAddress();
                        space = ", ";
                    }
                    response += "\r\n";
                }
```

```
                response += "Subject: " + message.getSubject() + "\r\n";

                response += "Date: " + message.getSentDate() + "\r\n";

                response += "Message-ID: " + message.getMessageID() + "\r\n";

                response += "Content-Type: " + message.getContentType() +
"\r\n";

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


        if (request.indexOf("BODY[]", separator)!=-1)
        {
            needbracket = false;
            String emlFile = "D:\\mail\\" + UID + ".eml";
            String strLine = null;
            String line = "";
            try{
                // Open the file that is the first
                // command line parameter
                FileInputStream fstream = new FileInputStream(emlFile);
                // Get the object of DataInputStream
                DataInputStream in = new DataInputStream(fstream);
                BufferedReader br = new BufferedReader(new
InputStreamReader(in));
                //Read File Line By Line

                while ((strLine = br.readLine()) != null)   {
                    line += strLine + "\n";
                    // Print the content on the console
                }
                //Close the input stream
                in.close();
            }catch (Exception e){//Catch exception if any
                System.err.println("Error: " + e.getMessage());
            }

            response += "BODY[]{" + line.length() + "}\n"  +line + "\n";


        }

        response += ")";

        SendAndPrint(response, out);

    }

    /**
     * Полученния envelop-контента сообщения
     * @param message - сообщения
     * @return - контент
     * @throws MessagingException - ошибка в сообщении
     */
    public String getEnvelop (MimeMessage message) throws MessagingException
    {
        String envelope ="ENVELOPE (\"" + message.getSentDate() + "\" " +
//date
                "\"" + message.getSubject() + "\" "; //subject
```

```
        //from
        InternetAddress[] adrFrom = (InternetAddress[]) message.getFrom();
        String from = parsAdressMass(adrFrom);
        envelope += from + " " ;

        //sender
        envelope += from + " " ;

        //reply-to
        InternetAddress[] adrReplyTo = (InternetAddress[])
message.getReplyTo();
        String replyTo = parsAdressMass(adrReplyTo);
        envelope += replyTo + " ";

        //to
        InternetAddress[] adrrTo = (InternetAddress[])
message.getRecipients(MimeMessage.RecipientType.TO);
        String to = parsAdressMass(adrrTo);
        envelope += to + " ";

        //cc
        InternetAddress[] adrrCc = (InternetAddress[])
message.getRecipients(MimeMessage.RecipientType.CC);
        String cc = parsAdressMass(adrrCc);
        if (cc!=null)
            envelope += cc ;
        else
            envelope += "NIL";

        //bcc
        InternetAddress[] adrrBcc = (InternetAddress[])
message.getRecipients(MimeMessage.RecipientType.BCC);
        String bcc = parsAdressMass(adrrBcc);
        if (bcc!=null)
            envelope +=  " " + bcc ;
        else
            envelope += " NIL";

        //in-reply-to??
        envelope += " NIL";

        envelope += " \"" + message.getMessageID() + "\")";
        return  envelope;
    }

    /**
     * Парсер адреса из InternetAddress в String
     * @param adr - адрес InternetAddress
     * @return - адрес String
     */
    public String parsAdress (InternetAddress adr )
    {
        String line = "";
        String person;
        String adress;
        String name;
        String host;

        try {
            person = adr.getPersonal();
            if (person == null)
                person = "NILL";
            else
                person = "\"" + person + "\"";
```

```
        } catch (NullPointerException e) {
            person = "NILL";
        }

        try {
            adress = adr.getAddress();
            adress = "\"" + adress + "\"";
        } catch (NullPointerException e) {
            adress = "NILL";
        }

        try {
            name = adress.substring(0,adress.indexOf('@'));
            name =  name + "\"";
        } catch (StringIndexOutOfBoundsException e) {
            name = "NILL";
        }

        try {
            host = adress.substring(adress.indexOf('@')+1,adress.length());
            host = "\"" + host;
        } catch (StringIndexOutOfBoundsException e) {
            host = "NILL";
        }

        line = "(" + person + " NILL " + name + " " + host + ")";

        return  line;
    }

    /**
     * Парсер массива адресов из InternetAddress в String
     * @param adr - массив адресов InternetAddress
     * @return - адреса String
     */
    public String parsAdressMass (InternetAddress[] adr )
    {
        String adressLine = "(";
        String from ="";
        try {
            if (adr.length >=0)
            {
                String line;
                for (int i = 0; i<adr.length;i++)
                {
                    from = parsAdress (adr[i]);
                    adressLine += from;
                }
            }
            else
                adressLine += "(NILL NILL NILL NILL)";
            adressLine += ")";
        } catch (NullPointerException e) //if empty field
        {
            return null;
        }

        return  adressLine;
    }

    /**
     * Получения тела сообщения
     * @param message - сообщение
     * @return - тело сообщения
```

```
     * @throws MessagingException - ошибка в письме
     * @throws IOException - ошибка работы с файлом
     */
    public String GetBodyStructure (MimeMessage message) throws
MessagingException, IOException
    {
        String contentType = message.getContentType();
        String ans = "BODY (\"";
        int nextIndex = 0;

        //get params
        for (int i = 0; i< contentType.length(); i++)
        {
            if (contentType.charAt(i) != '/')
            {
                if (contentType.charAt(i) ==';')
                {
                    i+= 2;
                    ans +="\"";
                    nextIndex = i;
                    break;
                }

                else
                    ans += contentType.charAt(i);
            }
            else
                ans +="\" \"";
        }

        ans+= " (\"";

        for (int i = nextIndex; i<contentType.length(); i++)
        {
            if (contentType.charAt(i) == '=')
                ans +="\" \"";
            else
                ans += contentType.charAt(i);
        }

        ans +="\") NIL NIL \"7BIT\" "; //7 BIT??

        int octetSize = (message.getSize() + 7)/8;
        //counting lines
        String mes = message.getContent().toString();
        int linesAmount = 0;
        for (int i=0; i<mes.length(); i++)
        {
            if (mes.indexOf('\n',i) >0)
            {
                i = mes.indexOf('\n',i);
                linesAmount ++;
            }
        }
        ans += octetSize + " " + linesAmount + ")";

        return ans;
    }
}
```

Класс CreateCommand

```
package Commands;

import DBHandler.DB;
```

```java
import Server.IncomingServer;

import java.io.PrintWriter;
import java.sql.ResultSet;
import java.sql.SQLException;


public class CreateCommand extends IncomingServer {
    public String prefix;
    public String request;
    public int UserId;
    public PrintWriter out;
    public DB assistant;

    public String unmarkedResponse;
    public String markedResponse;
    public String userId;
    public String folder;
    /**
     * Конструктор CreateCommand
     * @param host_port - порт на котором работает сервер
     */
    public CreateCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор CreateCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param UserId - ID пользователя
     * @param out - поток для ответа
     * @param assistant - объект для работы с бд
     */
    public CreateCommand (String prefix, String request, int UserId,
PrintWriter out, DB assistant) {//no-args constructor
        this.prefix = prefix;
        this.out = out;
        this.request = request;
        this.UserId = UserId;
        this.assistant = assistant;
    }

    /**
     * Выполнение комманды
     * Вначале извлекаем из запроса необходимые поля и далее обращаемся к дб
для создания папки.
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        userId = Integer.toString(UserId);
        folder = GetFirstParamFromRequest(request);

        Boolean create = CheckFolder();

        markedResponse = MakeMarked(create);

        SendAndPrint(markedResponse, out);
    }

    /**
     * Собираем строку ответа без префикса
     * @param create - true - если возможно создать папку
     *                 false - если такая папка уже есть
```

```
     * @return строка с ответом
     */
    public String MakeMarked (Boolean create) {
        String response = "";
        if (create)
        {
            int result =
assistant.AddFolderToUser(Integer.toString(UserId),folder);

            if (result == 1)
            {
                response = prefix + " OK CREATE completed";
            }
            else
            {
                response = prefix + " NO CREATE error";
            }
        }
        else
        {
            response =  prefix + " NO CREATE folder already exists";
        }

        return response;
    }

    /**
     * Проверка наличия папки
     * @return - true - если возможно создать папку
     *           false - если такая папка уже есть
     */
    public Boolean CheckFolder ()
    {
        Boolean create = null;
        ResultSet resultSet = assistant.SearchFolder(userId,folder);

        try {
            if (resultSet.next())    // if this folder already exists
            {
                create = false;
            }
            else
            {
                create = true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return create;
    }
}
```

Класс CopyCommand

```
package Commands;


import DBHandler.DB;
import Server.IncomingServer;

import java.io.PrintWriter;

public class CopyCommand extends IncomingServer {
```

```
    public String prefix;
    public String request;
    public DB assistant;
    public String unmarkedResponse;
    public String markedResponse;
    public PrintWriter out;
    /**
     * Конструктор CopyCommand
     * @param host_port - порт на котором работает сервер
     */
    public CopyCommand (int host_port) {//default constructor
        super(host_port);
    }

    /**
     * Конструктор CopyCommand
     * @param prefix - префикс команды
     * @param request - строка запроса
     * @param assistant - объект для работы с бд
     * @param out - поток для ответа
     */
    public CopyCommand (String prefix, String request, DB assistant,
PrintWriter out) {//no-args constructor
        this.prefix = prefix;
        this.request = request;
        this.assistant = assistant;
        this.out = out;
    }

    /**
     * Выполнение комманды
     * Вначале извлекаем из запроса необходимые поля и далее обращаемся к дб
для копирования сообщения.
     * Полученный результат отправляем клиенту
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        String UID = GetUID();
        String folder = GetFirstParamFromRequest(request);
        int result = CopyMessage(UID, folder);

        unmarkedResponse = MakeUnmarked(UID, folder);
        markedResponse = MakeMarked(result);

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);

    }

    /**
     * Получаем из строки запроса UID сообщения
     * @return UID
     */
    public String GetUID () {
        String UID ="";
        int startSymbol = request.indexOf('Y') + 2;
        int endSymbol = request.indexOf('\"')-2;

        System.out.println("start = " + startSymbol + " end = " + endSymbol);
        for (int i = startSymbol; i<= endSymbol; i++)
        {
            UID += request.charAt(i);
        }
        System.out.println("UID = " + UID);
```

```
        return UID;
    }

    /**
     * Отбращаемся к БД с запросом копирования сообщения в указаную папку
     * @param uid - UID сообщения для копирования
     * @param folder - папка назначения
     * @return 1 - если успешно
     */
    int CopyMessage (String uid, String folder)
    {
        int result = assistant.CopyMessage(uid, folder);

        return result;
    }

    /**
     * Собираем строку ответа без префикса
     * @param UID - UID сообщения для копирования
     * @param folder - папка назначения
     * @return строка с ответом
     */
    public String MakeUnmarked (String UID, String folder) {
        String response = prefix + " COPY " + UID + " \"" + folder + "\"";

        return response;
    }

    /**
     * Собираем строку ответа с префиксом
     * @param result - результат запроса к БД
     * @return строка с ответом
     */
    public String MakeMarked (int result) {
        String response = "";

        if (result == 1)
        {
            response = prefix + " OK COPY completed";
        }
        else
        {
            response = prefix + " NO [TRYCREATE] Folder doesn't exist";
        }

        return response;
    }
}
```

Класс CapabilityCommand

```
package Commands;


import Server.IncomingServer;

import java.io.PrintWriter;
public class CapabilityCommand extends IncomingServer {
    String prefix;
    PrintWriter out;
    public String unmarkedResponse;
    public String markedResponse;

    /**
     * Конструктор Capability
```

```
     * @param host_port - порт на котором работает сервер
     */
    public CapabilityCommand (int host_port) {
        super(host_port);
    }

    /**
     * Конструктор Capability
     * @param prefix - префикс команды
     * @param out - поток для ответа
     */
    public CapabilityCommand (String prefix, PrintWriter out) {
        this.prefix = prefix;
        this.out = out;
    }

    /**
     * Выполнение комманды
     * @see Server.IncomingServer#SendAndPrint(String, java.io.PrintWriter)
     */
    public void DoCommand () {
        unmarkedResponse = MakeUnmarked();
        markedResponse = MakeMarked();

        SendAndPrint(unmarkedResponse, out);
        SendAndPrint(markedResponse, out);
    }

    /**
     * Собираем строку ответа без префикса
     * @return строка с ответом
     */
    public String MakeUnmarked () {
        String response  =   "* CAPABILITY IMAP4rev1";

        return response;
    }

    /**
     * Собираем строку ответа с префиксом
     * @return строка с ответом
     */
    public String MakeMarked () {
         String response = prefix + " OK CAPABILITY completed";

        return response;
    }
}
```