

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчет о лабораторной работе

«Разработка клиента прикладного протокола POP3 на языке Java»

Дисциплина: «Сетевые технологии»

Выполнила студентка гр. 43501/3

_____ А.В. Емельянова
(подпись)

Руководитель

_____ К.Д. Вылегжанина
(подпись)

Санкт - Петербург

2016

1. Техническое задание.

Вариант задания: Клиент протокола POP3.

Разработать приложение для операционных систем семейства Windows или Linux, обеспечивающее функции клиента протокола POP-3.

1.1. Основные возможности:

- 1) Подключение к указанному серверу по IP-адресу или доменному имени
- 2) Получение состояния ящика (количество новых писем, их суммарная длина)
- 3) Получение списка заголовков всех новых писем сервера без предварительной загрузки
- 4) Загрузка всех новых или конкретных выбранных писем
- 5) Пометка конкретных писем для последующего удаления
- 6) Удаление помеченных писем
- 7) Выход из приложения без удаления помеченных писем
- 8) Подробное протоколирование соединения сервера с клиентом

1.2. Поддерживаемые команды протокола POP-3:

- USER – передача серверу идентификационной информации пользователя
- PASS – передача серверу пароля пользователя
- STAT – получение состояния почтового ящика
- LIST – получение списка сообщения почтового ящика
- RETR – получение сообщения
- DELE – пометка сообщения на удаление
- TOP – получение первых нескольких строк сообщения
- UIDL – получение уникального идентификатора сообщения
- RSET – сброс всех пометок на удаление сообщений
- QUIT – удаление всех помеченных сообщений и завершение сеанса

1.3. Настройки приложения.

Разработанное приложение должно предоставлять пользователю настройку следующих параметров:

- 1) IP-адрес или доменное имя почтового сервера
- 2) Номер порта сервера (по умолчанию - 110)
- 3) Имя пользователя
- 4) Пароль пользователя

1.4. Методика тестирования.

Для тестирования приложения следует использовать бесплатные почтовые серверы, имеющиеся в сети Internet.

Штатными клиентами электронной почты (Mozilla Thunderbird, MS Outlook Express, The Bat, Netscape Messenger) на сервер помещаются тестовые сообщения. В процессе тестирования проверяются основные возможности приложения.

2. Разработка приложения.

2.1. Протокол для реализации.

Действия, соответствующие каждой команде, указаны в п.1.2

Имя	Аргументы	Ограничения	Возможные ответы
USER	[имя]	—	* +OK name is a valid mailbox * -ERR never heard of mailbox name
PASS	[пароль]	Работает после успешной передачи имени почтового ящика	* +OK maildrop locked and ready * -ERR invalid password * -ERR unable to lock maildrop
DELE	[сообщение]	Доступна после успешной идентификации	* +OK message deleted * -ERR no such message
LIST	[сообщение]	Доступна после успешной идентификации	* +OK scan listing follows * -ERR no such message
RETR	[сообщение]	Доступна после успешной идентификации	* +OK message follows * -ERR no such message
RSET	—	Доступна после успешной идентификации	+OK
STAT	—	Доступна после успешной идентификации	+OK a b
TOP	[сообщение] [количество строк]	Доступна после успешной идентификации	* +OK n octets * -ERR no such message
QUIT	—	—	+OK

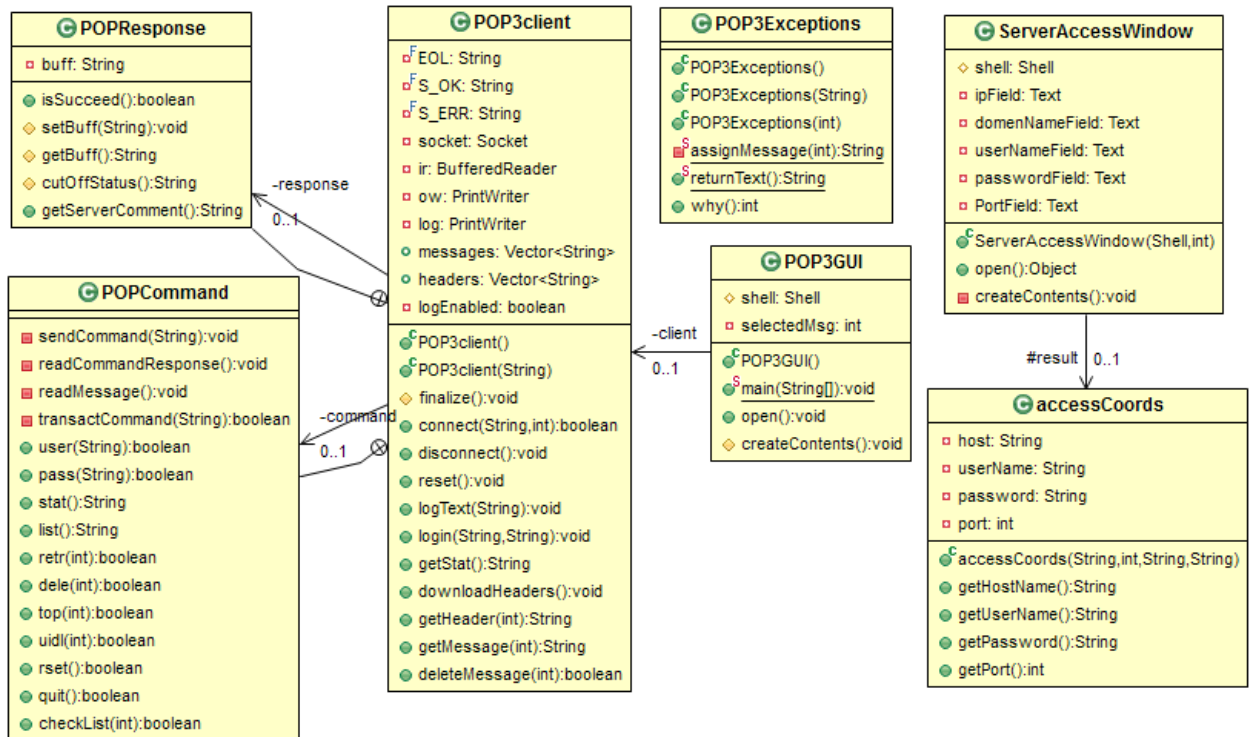
2.2. Архитектура приложения.

Большая часть функциональности реализована в классах POP3client, POPResponse и POPCommand.

Всё, что связано с графическим интерфейсом, содержится в классах POP3GUI (основное окно) и ServerAccessWindow (окно для установки соединения с сервером).









Для хранения и передачи настроек соединения используется класс accessCoords.

Для генерации исключений и соответствующих им сообщений об ошибках используется класс POP3Exceptions.



3. Тестирование и результаты работы приложения.

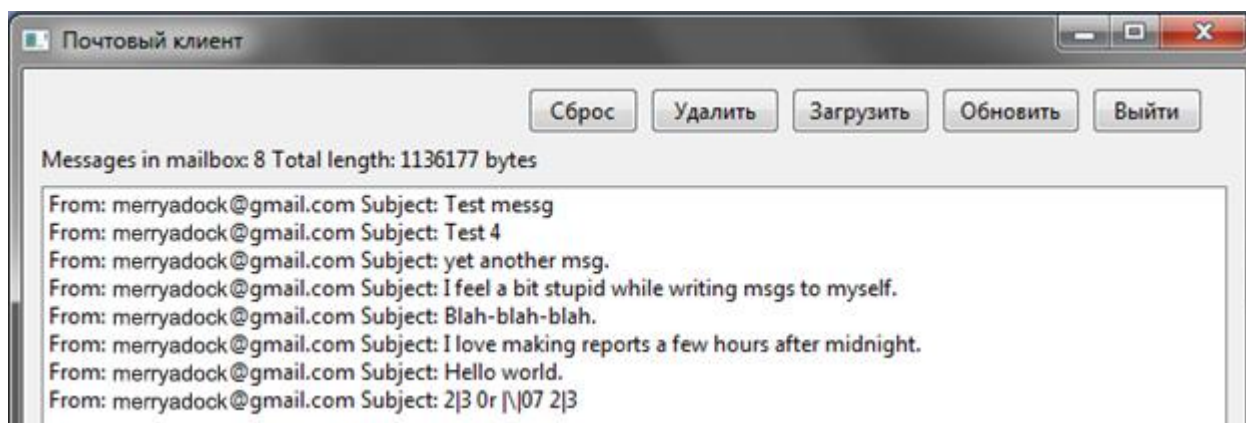
Для тестирования будем использовать почтовый ящик на mail.ru. С другого почтового ящика отправим несколько сообщений.

<input type="checkbox"/>	 Anastasia Yemeliano'	2 3 0r \\07 2 3 50 V 3 v3r/' \\73r357 n9 73x7... 4(7u4 /, 4b50 u73 /' n07.
<input type="checkbox"/>	 Anastasia Yemeliano'	Hello world. Hello world.Hello world.Hello world.Hello world.Hello world.
<input type="checkbox"/>	 Anastasia Yemeliano'	I love making reports a few hours after midnight. I love making reports a few hours after
<input type="checkbox"/>	 Anastasia Yemeliano'	Blah-blah-blah. Blah-blah-blah. Blah-blah-blah. Blah-blah-blah. Blah-blah-bl:
<input type="checkbox"/>	 Anastasia Yemeliano'	I feel a bit stupid while writing msgs to myself. Some text.
<input type="checkbox"/>	 Anastasia Yemelianova	yet another msg. At half-past twelve next day Lord Henry Wotton strolled from Curzon Stre
<input type="checkbox"/>	 Anastasia Yemelianova	Test4 Gergthrtrhttrhttrhttrhttrhtth weGergthrtrhttrhttrhttrhttrhtth weGergthrtrhttrhttrhttrhttr
<input type="checkbox"/>	 Anastasia Yemelianova	Test messg Qedegthrth trhttrhttr ht thtrhttrht thrttrhttrht thrttrhttrht rt rth rth rthtrhttrhttrht

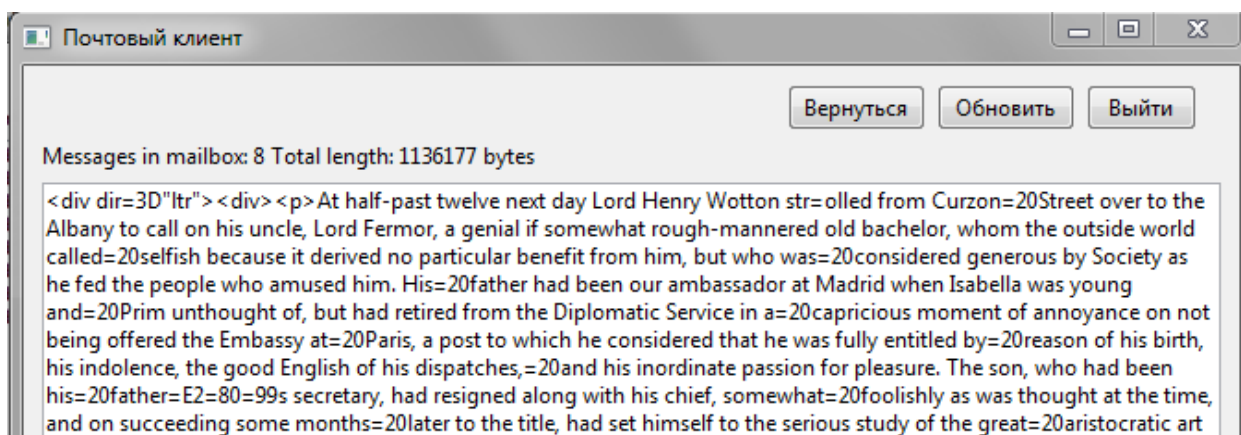
Установим соединение с сервером посредством данной формы:

IP-адрес	<input type="text"/>
Доменное имя	<input type="text" value="pop.mail.ru"/>
Номер порта	<input type="text" value="995"/>
Имя пользователя	<input type="text"/>
Пароль	<input type="password" value="....."/>
<input type="button" value="Подключиться"/>	

После успешной авторизации видим:

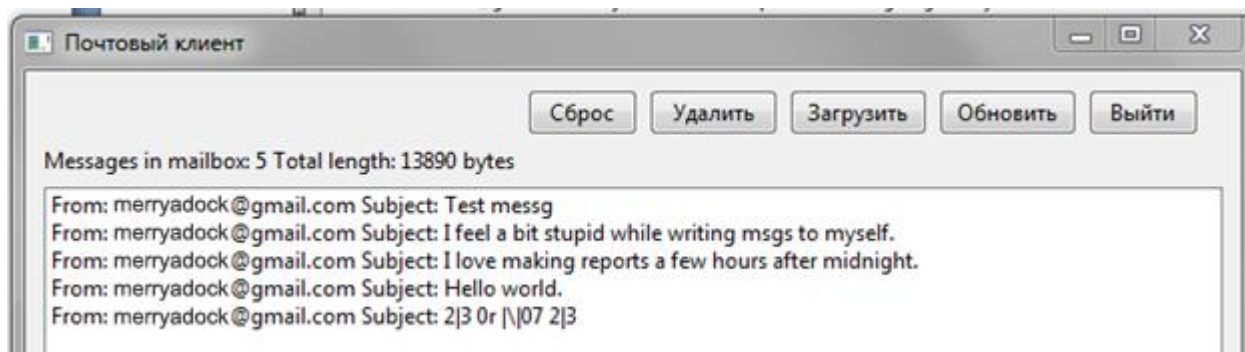


Загрузим одно из сообщений:



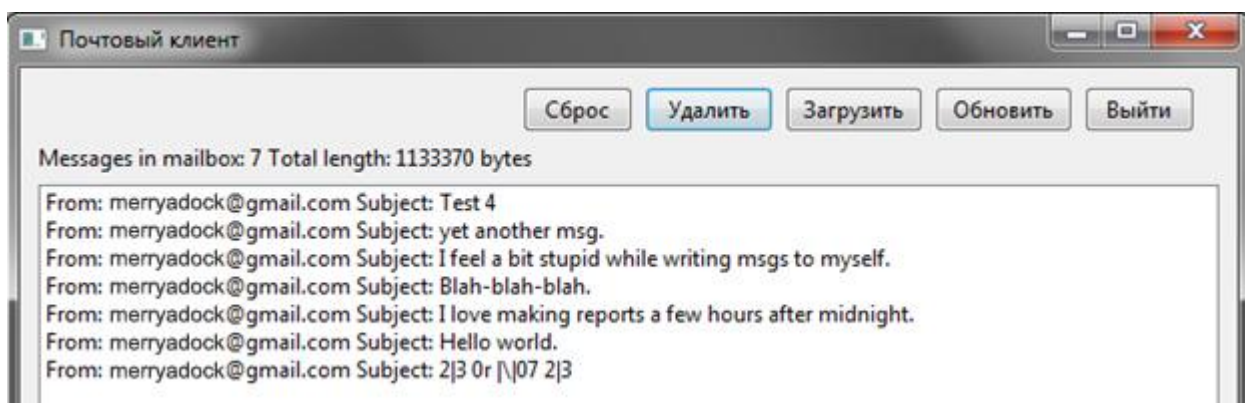
Также, убедимся, что приложение может работать с письмами больших размеров. Для этого отправим письмо с текстом весом порядка 2Мб, и откроем его в приложении. Увидим, что есть возможность увидеть письмо целиком.

Удалим несколько сообщений:



Откатим транзакцию (кнопка «сброс»), после чего увидим, что все удаленные сообщения восстановлены.

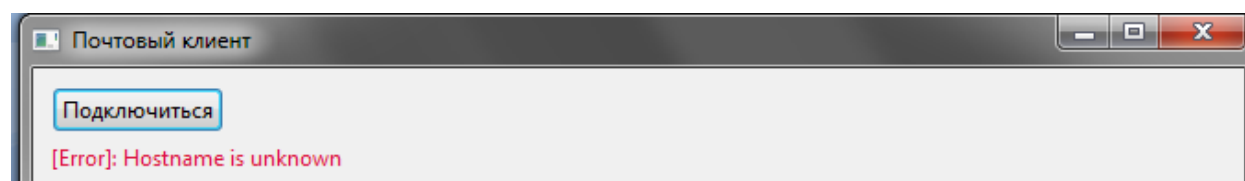
Теперь удалим первое сообщение:



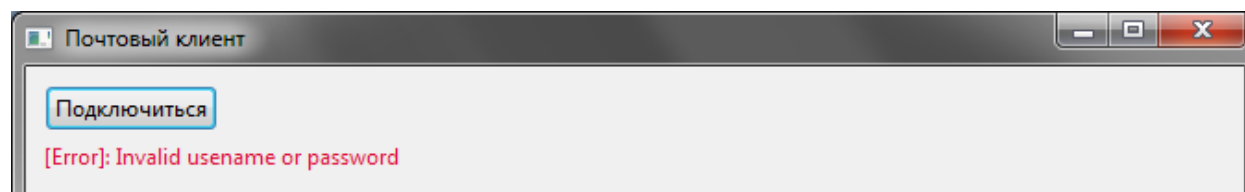
И нажмем на кнопку «выйти», что завершит транзакцию. Открыв почтовый ящик через веб-интерфейс убедимся, что письмо действительно удалено.

Проверим также вывод ошибок при авторизации.

Введем случайный набор символов вместо адреса хоста:



Случайный набор символов вместо e-mail адреса:



Проведя ещё ряд манипуляций с удалением, открытием и восстановлением сообщений убедимся, что приложение работает верно. Проверив лог, убедимся, что все выполненные действия там отражены.

4. Выводы.

Язык программирования Java является удобным инструментом для разработки сетевых приложений с использованием стека протоколов TCP/IP, так как содержит стандартные компоненты, обеспечивающие простоту работы с данными протоколами, по сравнению с разработкой на языках C/C++.

В результате работы было разработано приложение с графическим интерфейсом, обеспечивающее основные функции клиента протокола POP3.

Так как данная реализация клиента тестировалась с использованием подключения к почтовому серверу mail.ru, потребовалось использовать шифрование с использованием SSL. Данная функция реализована с помощью библиотеки javax.net.ssl. В целом, стоит отметить большое количество существующих библиотек в Java, с помощью которых можно было бы ещё больше упростить процесс разработки.

Приложение 1. Класс POP3Exceptions

```
public class POP3Exceptions extends Exception {
    private static final long serialVersionUID = 6942766943501579458L;
    // Types of situations
    public static final int NOT_AVAILABLE = 0;
    public static final int BAD_RESPONSE = 1;
    public static final int BAD_NAME = 2;
    public static final int BAD_PASSWORD = 3;
    public static final int SOCKET_ERROR = 4;
    public static final int HOST_UNKNOWN = 5;
    public static final int IO_ERROR = 6;
    public static final int RETR_ERROR = 7;

    //Reason of exception
    private static int why = NOT_AVAILABLE;

    public POP3Exceptions(){
        super();
    }

    public POP3Exceptions(String message){
        super(message);
    }

    public POP3Exceptions(int reason){
        super(POP3Exceptions.assignMessage(reason));
    }

    //Choose string to return err message
    private static String assignMessage(int reason) {
        why = reason;

        return returnText();
    }

    public static String returnText()
    {
        int reason = why;
        switch(reason)
        {
            case BAD_RESPONSE:
                return new String("Bad response from the mail server");
            case SOCKET_ERROR:
                return new String("Socket error occurred");
            case BAD_NAME:
                return new String("User with this name not found");
            case BAD_PASSWORD:
                return new String("Invalid username or password");
            case HOST_UNKNOWN:
                return new String("Hostname is unknown");
            case IO_ERROR:
                return new String("I/O error");
            case RETR_ERROR:
                return new String("Fatal error occured during message reading");
            default:
                return new String("Unknown failure. Sorry. We really tried");
        }
    }

    public int why()
    {
        return why;
    }
}
```


Приложение 2. Класс POP3client

```
public class POP3client
{
    // MODE
    private final static boolean debug = true;

    // CONSTANTS
    private final String EOL = "";
    private final String S_OK = "+OK";
    private final String S_ERR = "-ERR";

    // Class fields
    private POPCommand command = null;
    private POPResponse response = null;
    private Socket socket = null;
    private BufferedReader ir = null;
    private PrintWriter ow = null;
    private PrintWriter log = null;
    public Vector<String> messages = null;
    public Vector<String> headers = null;
    private boolean logEnabled = false;

    public POP3client()
    {
        command = this.new POPCommand();
        response = this.new POPResponse();
        messages = new Vector<String>();
        headers = new Vector<String>();
    }

    public POP3client(String logName)
    {
        this();
        logEnabled = true;
        try
        {
            log = new PrintWriter(new FileOutputStream(logName, true), true);
        }
        catch(IOException e)
        {
            // Problem with creating protocol file
            logEnabled = false;
        }
    }

    // If user forgot to disconnect
    protected void finalize() throws Throwable
    {
        reset();
        disconnect();
    }

    // Set connection
    public boolean connect(String hostName, int portNumber) throws POP3Exceptions
    {
        try
        {
            logText("Creating a socket...");
            SSLSocketFactory fac = (SSLSocketFactory) SSLSocketFactory.getDefault();
            socket = fac.createSocket(hostName, portNumber);

            logText("Creating an input stream...");
            ir = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            logText("Creating an output stream...");
            ow = new PrintWriter(new DataOutputStream(socket.getOutputStream()), true);
        }
    }
}
```

```

        command.readCommandResponse();
        return response.isSuccessed();
    }
    catch(UnknownHostException e)
    {
        logText("Host unknown");
        throw new POP3Exceptions(POP3Exceptions.HOST_UNKNOWN);
    }
    catch(IOException e)
    {
        logText("Creating an I/O channel failed");
        throw new POP3Exceptions(POP3Exceptions.SOCKET_ERROR);
    }
}

public void disconnect()
{
    try
    {
        logText("Disconnecting... ");
        // Send request for disconnection
        command.quit();
        if(ir != null)
        {
            ir.close();
            ir = null;
        }
        logText("[Input stream closed]");
        if(ow != null)
        {
            ow.close();
            ow = null;
        }
        logText("[Output stream closed]");
        if(socket != null)
        {
            socket.close();
            socket = null;
        }
        logText("[Socket closed]");
        logText("[Disconnected]");
        if(log != null)
        {
            log.close();
            log = null;
        }
    }
    catch(Exception e)
    { logText("Disconnection failed!");}
}

public void reset() throws POP3Exceptions
{
    logText("Cancelling current transaction...");
    command.rset();
    downloadHeaders();
}

public void logText(String text)
{
    if(logEnabled) log.println(text);
    if(debug) System.out.println(text);
}

public void login(String name, String password) throws POP3Exceptions
{
    logText("Sending the user name...");

```

```

        if( !command.user(name) )
            throw new POP3Exceptions(POP3Exceptions.BAD_NAME);
        logText("Sending the password...");
        if(password != null)
            if( !command.pass(password) )
                throw new POP3Exceptions(POP3Exceptions.BAD_PASSWORD);
    }

    //Getting mailbox state
    public String getStat() throws POP3Exceptions
    {
        String statResult = command.stat();
        String tmp[] = statResult.split(" ");
        int countOfMessages = Integer.parseInt(tmp[1]);
        int sizeOfAllMessages = Integer.parseInt(tmp[2]);
        return "Messages in mailbox: "+countOfMessages+" Total length: "+sizeOfAllMessages+"
bytes";
    }

    /**
     * Downloading all headers, saves in into headers array
     * */
    public void downloadHeaders() throws POP3Exceptions
    {
        headers.clear();
        int tmpCnt=0;
        int i;
        for(i=1; tmpCnt<4; i++)
        {
            if(!command.checkList(i))
                tmpCnt++;
            else if(command.top(i))
            {
                int offset = response.buff.indexOf(EOL);
                headers.addElement(response.buff.substring(offset + EOL.length()));
                logText("Received header:\n"+response.buff.substring(offset +
EOL.length()));
            }
            else throw new POP3Exceptions(POP3Exceptions.RETR_ERROR);
        }
    }

    /**
     * Take (from headers array) and (trying to) parse header.
     * Before calling it you should run downloadHeaders() method
     * @return String in format: From: (e-mail address) Subject: (subject)
     * @param number - number of message to get header (number>0)
     * */
    public String getHeader(int number)
    {
        if(headers.size()<1 || headers.size()<number)
        {
            logText("Wrong number of letter to download: "+number);
            return "Error";
        }
        String tmp = (String)headers.elementAt(number-1);
        tmp = tmp.substring(tmp.indexOf("From: ")+1);
        tmp = tmp.substring(tmp.indexOf("<")+1);
        String result = "From: "+tmp.substring(0,tmp.indexOf(">"));
        tmp = tmp.substring(tmp.indexOf("Subject: ")+9);
        tmp = tmp.substring(0,tmp.indexOf("To: "));
        return result+" Subject: "+tmp;
    }

    /**
     * Download from server and (trying to) parse header.

```

```

* Should show html-version (with tags, yep =_)
* I swear, it works with current mail.ru and gmail.com versions of letters
* @return String in format: From: (e-mail address) Subject: (subject)
* @param number - number of message to get header (number>0)
* */
public String getMessage(int number) throws POP3Exceptions
{
    String tmp="";
    if(command.retr(number)){
        int offset = response.buff.indexOf(EOL);
        tmp = response.buff.substring(offset + EOL.length());
        tmp = tmp.substring(tmp.indexOf("Content-Type: text/html;")+25);
        tmp = tmp.substring(tmp.indexOf("<"));
        tmp = tmp.substring(0, tmp.lastIndexOf("--"));
        tmp = tmp.substring(0, tmp.lastIndexOf("--"));

        messages.addElement(tmp);
        logText(tmp);
    }
    return tmp;
}

public boolean deleteMessage(int number) throws POP3Exceptions
{
    if(command.dele(number))
    {
        downloadHeaders();
        return true;
    }
    return false;
}

```

Приложение 3. Класс POPResponse

```

class POPResponse
{
    private String buff = "";

    /*
    * True if last command executed successful
    * False if not
    * Throws exception if server return smth strange
    * */
    public boolean isSucceed() throws POP3Exceptions
    {
        boolean result = true;

        if(!buff.startsWith(S_OK))
        {
            if(!buff.startsWith(S_ERR))
            {
                throw new POP3Exceptions(POP3Exceptions.BAD_RESPONSE);
            }
            result = false;
        }
        return result;
    }

    //Save data in temporary buffer
    protected void setBuff(String s)
    {
        buff = s;
    }

    protected String getBuff()
    {
        return buff;
    }
}

```

```

    }

    protected String cutOffStatus()
    {
        int offset = buff.indexOf(' ');
        if(offset != -1)
        {
            String tmpStr = buff.substring(offset);
            return tmpStr.trim();
        }
        return null;
    }

    public String getServerComment()
    {
        String tmpStr = null;

        tmpStr = cutOffStatus();
        int offset = tmpStr.indexOf(EOL);
        if(offset != -1)
            return tmpStr.substring(0, offset);
        return null;
    }
}

```

Приложение 4. Класс POPCommand

```

class POPCommand
{
    private void sendCommand(String command) throws POP3Exceptions
    {
        logText("Sending command... ");
        ow.println(command);
    }

    private void readCommandResponse() throws POP3Exceptions
    {
        logText("Reading response...");
        StringBuffer tmpBuff = new StringBuffer();

        try { tmpBuff.append(ir.readLine()); }
        catch(IOException e)
        {
            throw new POP3Exceptions(POP3Exceptions.IO_ERROR);
        }
        response.setBuff(tmpBuff.toString());
    }

    private void readMessage() throws POP3Exceptions
    {
        logText("Reading message...");
        String tmpStr = new String("");
        StringBuffer tmpBuff = new StringBuffer();

        try
        {
            //Read strings until terminator is found
            while(!(tmpStr = ir.readLine()).equals("."))
                tmpBuff.append(tmpStr + "");
        }
        catch(IOException e)
        {
            throw new POP3Exceptions(POP3Exceptions.IO_ERROR);
        }
        tmpStr = tmpBuff.toString();
        response.setBuff(tmpStr);
    }
}

```

```

// Do typical command
private boolean transactCommand(String command) throws POP3Exceptions
{
    sendCommand(command);
    readCommandResponse();
    return response.isSuccess();
}

/*
 * MAIN COMMANDS OF POP3 PROTOCOL
 */

// Sends user information to server
public boolean user(String name) throws POP3Exceptions
{
    return transactCommand("USER " + name);
}

// Sends password
public boolean pass(String password) throws POP3Exceptions
{
    return transactCommand("PASS " + password);
}

// Getting mailbox state
public String stat() throws POP3Exceptions
{
    sendCommand("STAT");
    readCommandResponse();
    return response.buff;
}

// Get list
public String list() throws POP3Exceptions
{
    sendCommand("LIST");
    readCommandResponse();
    return response.buff;
}

// Read message
public boolean retr(int number) throws POP3Exceptions
{
    if(number != 0)
    {
        sendCommand("RETR " + Integer.toString(number));
        readMessage();
        return response.isSuccess();
    }
    else return false;
}

// Select message to delete
public boolean dele(int number) throws POP3Exceptions
{
    if(number != 0)
        return transactCommand("DELE " + Integer.toString(number));
    else return false;
}

// Read few lines of message
public boolean top(int number) throws POP3Exceptions
{
    if(number != 0)
    {
        sendCommand("TOP " + Integer.toString(number)+" 2");
        readMessage();
    }
}

```

```

        return response.isSuccess();
    }
    else return false;
}

// Read message identifier
public boolean uidl(int number) throws POP3Exceptions
{
    if(number != 0)
    {
        return transactCommand("UIDL " + Integer.toString(number));
    }
    else return false;
}

// Cancel transaction
public boolean rset() throws POP3Exceptions
{
    return transactCommand("RSET");
}

// Delete all selected messages and quit
public boolean quit() throws POP3Exceptions
{
    return transactCommand("QUIT");
}

// Check
public boolean checkList(int number) throws POP3Exceptions
{
    if(number != 0)
        return transactCommand("LIST " + Integer.toString(number));
    else return false;
}
}
}

```

Приложение 5. Класс POP3GUI

```

public class POP3GUI {
    protected Shell shell;
    private POP3client client;
    private int selectedMsg=0;

    /**
     * Launch the application.
     * @param args
     */
    public static void main(String[] args) {
        try {
            POP3GUI window = new POP3GUI();
            window.open();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Open the window.
     */
    public void open() {
        Display display = Display.getDefault();
        createContents();
        shell.open();
        shell.layout();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
    }
}

```



```

cancelButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        try {
            client.reset();
        } catch (POP3Exceptions e1) {
            lblState.setVisible(false);
            lblError.setVisible(true);
            lblError.setText("[Error]: "+e1.returnText());
            return;
        }
    }
});

updButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        lblError.setVisible(false);
        lblError.setVisible(false);
        try {
            lblState.setText(client.getStat());
            lblState.setVisible(true);
            client.downloadHeaders();

            list.removeAll();
            for(int i = 1; i<=client.headers.size(); i++){
                list.add(client.getHeader(i));
            }
        } catch (POP3Exceptions e1) {
            lblError.setVisible(true);
            lblError.setText("[Error]: "+e1.returnText());
            return;
        }
    }
});

deleteButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        selectedMsg = list.getSelectionIndex()+1;
        if(selectedMsg>0){
            try {
                client.deleteMessage(selectedMsg);
                lblState.setText(client.getStat());
                lblState.setVisible(true);
                client.downloadHeaders();
                list.removeAll();
                for(int i = 1; i<=client.headers.size(); i++){
                    list.add(client.getHeader(i));
                }
            } catch (POP3Exceptions e1) {
                lblState.setVisible(false);
                lblError.setVisible(true);
                lblError.setText("[Error]: "+e1.returnText());
                return;
            }
        }
    }
});

disconnectButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        client.disconnect();
        connectButton.setVisible(true);
        disconnectButton.setVisible(false);
    }
});

```

```

        updButton.setVisible(false);
        list.removeAll();
        list.setVisible(false);
        lblState.setVisible(false);
        downloadButton.setVisible(false);
        cancelButton.setVisible(false);
        deleteButton.setVisible(false);
        styledText.setVisible(false);
        returnButton.setVisible(false);
    }
});

connectButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        client = new POP3client("logPOP3.txt");
        ServerAccessWindow dialogue = new ServerAccessWindow(shell, 0);
        dialogue.open();
        //here we have some result from our dialogue

        lblError.setVisible(false);
        lblState.setVisible(false);
        //setting connection
        try {
            client.connect(dialogue.result.getHostName(),
dialogue.result.getPort());
            client.login(dialogue.result.getUserName(),
dialogue.result.getPassword());

            lblState.setText(client.getStat());
            lblState.setVisible(true);
            client.downloadHeaders();

            for(int i = 1; i<=client.headers.size() && i<28; i++){
                list.add(client.getHeader(i));
            }

        } catch (POP3Exceptions e1) {
            lblError.setVisible(true);
            lblError.setText("[Error]: "+e1.returnText());
            return;
        }
        list.setVisible(true);
        connectButton.setVisible(false);
        disconnectButton.setVisible(true);
        updButton.setVisible(true);
        downloadButton.setVisible(true);
        deleteButton.setVisible(true);
        cancelButton.setVisible(true);
    }
});

downloadButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        selectedMsg = list.getSelectionIndex()+1;
        if(selectedMsg>0){
            try {
                list.setVisible(false);
                styledText.setVisible(true);
                styledText.setText(client.getMessage(selectedMsg));
                downloadButton.setVisible(false);
                returnButton.setVisible(true);
                deleteButton.setVisible(false);
                cancelButton.setVisible(false);
            } catch (POP3Exceptions e1) {

```

```

        lblState.setVisible(false);
        lblError.setVisible(true);
        lblError.setText("[Error]: "+e1.returnText());
        return;
    }
}

});

returnButton.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        returnButton.setVisible(false);
        downloadButton.setVisible(true);
        styledText.setVisible(false);
        list.setVisible(true);
        deleteButton.setVisible(true);
        cancelButton.setVisible(true);
    }
});
}
}

```