

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра теоретической прикладной информатики

Расчетно-графическое задание по дисциплине «Операционные системы, среды и оболочки»

Факультет:	ПМИ
Группа:	ПМ-92
Бригада:	9
Вариант:	12
Студенты:	Иванов В., Кутузов И.
Преподаватель:	Сивак М.А.

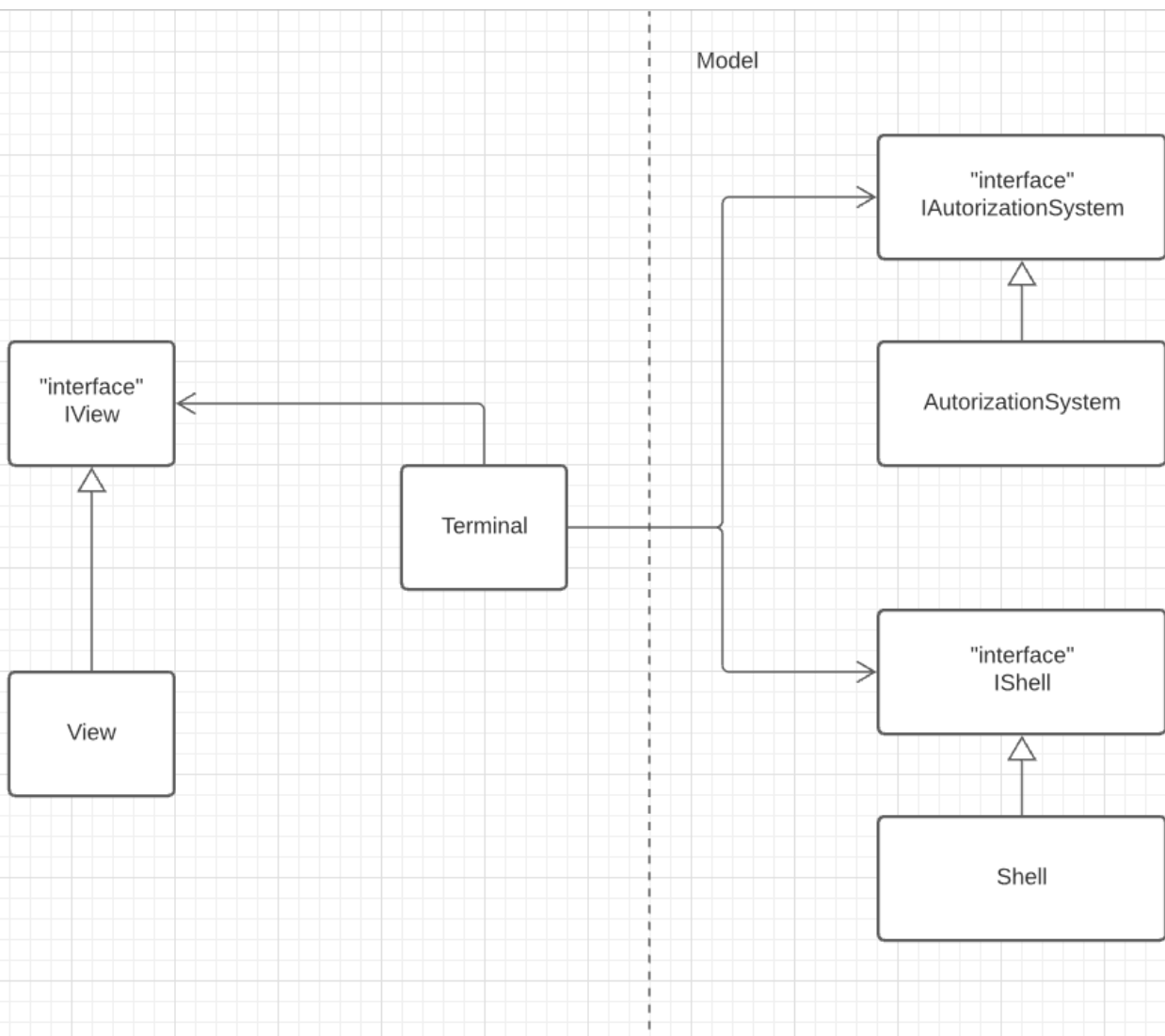
Новосибирск

2021

Описание работы программы

В основе программы лежит использование паттерна MVP. Программу условно можно разделить на три части:

- **Модель** (Authorization system и Command Shell) – хранит команды и данные о зарегистрированных пользователях, также отвечает за выполнение команд.
- **Вид** (View) – предоставляет пользователю интерфейс управления, а также отображает данные, полученные из модели.
- **Представитель** (Terminal) – является связующим звеном между моделью и видом.



Терминал

Рассмотрим подробнее работу **Terminal**.

```
public class Terminal
{
    private IView _view;
    private IShell _shell;
    private IAuthorizationSystem _authorization;
    private ITextHandler<Command> _commandTextHandler;

    public Terminal(IView view, IShell shell)
    {
        _view = view;
        _shell = shell;
        _authorization = new AuthorizationSystem();
        _commandTextHandler = new CommandTextHandler();
        Logout(_view, EventArgs.Empty);
    }

    private void HandleCommand(object sender, EventArgs e)
    {
        string content = _view.GetContentFromCommandLine();
        Command command = _commandTextHandler.Convert(content);

        _view.PrintContent(_view.GetStatus() + content);
        _view.PrintContent(_shell.ExecuteCommand(command.Name,
command.Arguments));
        _view.ChangeStatus(Directory.GetCurrentDirectory());
    }

    private void TryLogin(object sender, EventArgs e)
    {
        User user = new User(_view.GetLogin(), _view.GetPassword());

        if (_authorization.IsExistUser(user.Name, user.Password))
        {
            _view.EnterDown -= Login;
            _view.EnterDown += HandleCommand;
            _view.CtrlDDown += Logout;
            _view.ClearAuthorizationField();
            _view.ChangeStatus(Directory.GetCurrentDirectory());
            _view.ShowMainField();
        }
        _view.ClearAuthorizationField();
    }

    private void Logout(object sender, EventArgs e)
    {
        _view.ClearMainField();
        _view.EnterDown -= HandleCommand;
        _view.CtrlDDown -= Logout;
        _view.EnterDown += Login;
        _view.ShowAuthorizationField();
    }
}
```

Terminal реагирует на события, вызванные пользователем во время работы с интерфейсом программы, запрашивает данные из **View** и в зависимости от своего состояния (когда пользователь не авторизован или когда пользователь авторизован) отправляет эти данные на обработку в **Command shell** или **Authorization system**. Рассмотрим каждый случай в отдельности.

Пользователь не авторизован. Допустим пользователь набирает какие-то данные и нажимает на клавишу “Enter”, тем самым вызывая событие “EnterDown”, которое обрабатывается **Terminal** таким образом:

```
private void TryLogin(object sender, EventArgs e)
{
    User user = new User(_view.GetLogin(), _view.GetPassword());

    if (_authorization.IsExistUser(user.Name, user.Password))
    {
        _view.EnterDown -= Login;
        _view.EnterDown += HandleCommand;
        _view.CtrlDDown += Logout;
        _view.ClearAuthorizationField();
        _view.ChangeStatus(Directory.GetCurrentDirectory());
        _view.ShowMainField();
    }
    _view.ClearAuthorizationField();
}
```

Terminal делает запрос на получение данных, введенных пользователем, затем отправляет эти данные в **Authorization system**, если **Authorization system** подтверждает наличие зарегистрированного пользователя в своей базе данных, в таком случае **Terminal** меняет свое состояние, в котором может обрабатывать команды пользователя, а также меняет интерфейс **View**, чтобы пользователь имел возможность вводить команды.

Пользователь авторизован. Теперь пользователь решил ввести команду и отправить ее на обработку программе.

```
private void HandleCommand(object sender, EventArgs e)
{
    string content = _view.GetContentFromCommandLine();
    Command command = _commandTextHandler.Convert(content);

    _view.PrintContent(_view.GetStatus() + content);
    _view.PrintContent(_shell.ExecuteCommand(command.Name,
command.Arguments));
    _view.ChangeStatus(Directory.GetCurrentDirectory());
}
```

Terminal работает аналогично предыдущему случаю, все также запрашивает данные и отправляет их в **Command shell** для того, чтобы она выполнила команду, введенную пользователем.

Из этого состояния также можно перейти в состояние, когда пользователь не авторизован:

```
private void Logout(object sender, EventArgs e)
{
    _view.ClearMainField();
    _view.EnterDown -= HandleCommand;
    _view.CtrlDDown -= Logout;
    _view.EnterDown += Login;
    _view.ShowAuthorizationField();
}
}
```

Система авторизации

Authorization system содержит в себе “базу данных” зарегистрированных пользователей и может ответить на вопрос – “существует ли пользователь в базе?”

```
public interface IAuthorizationSystem
{
    bool IsExistUser(string userName, string password);
}

public class AuthorizationSystem : IAuthorizationSystem
{
    private IDatabase<IUser> _userDataBase;

    public AuthorizationSystem()
    {
        _userDataBase = new UserDataBase();
    }

    public bool IsExistUser(string userName, string password)
    {
        var users = _userDataBase.GetData();
        foreach(IUser user in users)
        {
            if(userName == user.Name && password == user.Password)
            {
                return true;
            }
        }
        return false;
    }
}
```

User data base представляет собой статическую базу данных, т.е. добавить нового пользователя или удалить старого возможности нет. Она соответственно хранит данные о пользователях и предоставляет их в режиме для чтения.

```
public class UserDataBase : IDatabase<IUser>
{
    private List<IUser> _userList;

    public UserDataBase()
    {
        _userList = new List<IUser>();
        _userList.Add(new Root());
        _userList.Add(new Hunky());
        //Add more users
    }

    public IReadOnlyList<IUser> GetData()
    {
        return _userList.AsReadOnly();
    }
}
```

User инкапсулирует в себе данные о пользователе.

```
public interface IUser
{
    string Name { get; }
    string Password { get; }
}
```

Командная оболочка

Command shell содержит в себе базу данных команд и исполняет их. Если запрашиваемая команда содержится в базе данных, тогда, соответственно, она же и исполняется, если же команды в базе данных не существует, тогда выполняется “несуществующая команда”.

```
public interface IShell
{
    string ExecuteCommand(string commandName, string[] arguments);
}

public class Shell : IShell
{
    private IDatabase<ICommand> _commandDataBase;

    public Shell()
    {
        _commandDataBase = new CommandDataBase();
    }

    public string ExecuteCommand(string commandName, string[] arguments)
    {
        var commands = _commandDataBase.GetData();

        foreach (ICommand command in commands)
        {
            if (command.Name == commandName)
            {
                return command.Execute(arguments);
            }
        }
        return NonExistentCommand.Execute();
    }
}
```


Command data base хранит в себе команды и предоставляет их список в режиме для чтения.

```
public class CommandDataBase : IDatabase<ICommand>
{
    private List<ICommand> _commandList;

    public CommandDataBase ()
    {
        _commandList = new List<ICommand>();
        _commandList.Add(new PrintWorkingDirectory());
        _commandList.Add(new FileList());
        _commandList.Add(new ChangeDirectory());
        _commandList.Add(new Concatenate());
        _commandList.Add(new MakeDirectory());
        _commandList.Add(new UnixName());
        _commandList.Add(new RemoveDirectory());
        _commandList.Add(new Remove());
        _commandList.Add(new FileType());
        _commandList.Add(new ShowDate());
        //Add more commands
    }

    public IReadOnlyList<ICommand> GetData ()
    {
        return _commandList.AsReadOnly();
    }
}
```

Command представляет собой команду, которая имеет имя и может исполняться.

```
public interface ICommand
{
    string Name { get; }
    string Execute(string[] arguments);
}
```

Non-existent command (“не существующая команда”) в сущности не является командой (т.к не реализует интерфейс **ICommand**), нужна лишь только для того, чтобы сохранить логику в коде.

```
public static class NonExistentCommand
{
    public static string Execute()
    {
        return "no such command exists";
    }
}
```

Чтобы объединить команды в единую абстрактную сущность было принято решение, что любая команда возвращает строку, как результат исполнения; любая команда принимает массив строк в качестве аргумента; ключ также является аргументом команды и в командной строке следует строго за именем через “пробел” (ключ все также является необязательным параметром). За счет этого также все данные о конкретной команде будут инкапсулированы в ней самой, т.е команда сама следит за аргументами переданными ей и определяет как с ними работать, таким образом не придется следить за аргументами “снаружи”.

Рассмотрим в качестве примера команду “uname”.

```
public static class BasicSystemInfo
{
    public static string ToString()
    {
        StringBuilder output = new StringBuilder();
        output.AppendFormat("Windows version : {0}\r\n",
Environment.OSVersion);
        output.AppendFormat("64 Bit operating system : {0} \r\n",
Environment.Is64BitOperatingSystem? "Yes" : "No");
        output.AppendFormat("Computer name : {0} \r\n",
Environment.MachineName);
        output.AppendFormat("Processor count : {0} \r\n",
Environment.ProcessorCount);
        output.AppendFormat("System directory : {0} \r\n",
Environment.SystemDirectory);
        output.AppendFormat("Logical drives : {0} \r\n",
            String.Join(", ", Environment.GetLogicalDrives())
                .Replace("\\", String.Empty));
        return output.ToString();
    }
}

class UnixName : ICommand
{
    public string Name => "uname";

    public string Execute(string[] arguments)
    {
        if(arguments == null)
        {
            return BasicSystemInfo.ToString();
        }
        else
        {
            if (arguments[0] == "-o")
            {
                return Environment.OSVersion.ToString();
            }
            if(arguments[0] == "-m")
            {
                return Environment.MachineName;
            }
        }
        return "the number of arguments does not match the command";
    }
}
```

Вид

View предоставляет пользователю интерфейс управления, а также отображает данные, полученные из модели.

```
public interface IView
{
    void ClearAuthorizationField();
    void ClearMainField();
    void ShowAuthorizationField();
    void ShowMainField();
    void PrintContent(string content);
    string GetContentFromCommandLine();
    string GetLogin();
    string GetPassword();
    void ChangeStatus(string status);
    string GetStatus();

    event EventHandler EnterDown;
    event EventHandler CtrlDDown;
}

public partial class View : Form, IView
{
    public View()
    {
        InitializeComponent();
    }

    public string GetContentFromCommandLine()
    {
        var content = inputField.GetContent();
        inputField.Clear();
        return content;
    }

    public void PrintContent(string content)
    {
        outputField.PrintText(content);
    }

    public void ChangeStatus(string status)
    {
        this.status.Change(status);
    }

    public string GetStatus()
    {
        return status.Get();
    }

    public void ShowAuthorizationField()
    {
        authorizationField.Visible = true;
        status.Visible = false;
        inputField.Visible = false;
        outputField.Visible = false;
    }
}
```

```

public void ShowMainField()
{
    authorizationField.Visible = false;
    status.Visible = true;
    inputField.Visible = true;
    outputField.Visible = true;
    inputField.Select();
}

public string GetLogin() => authorizationField.GetLogin();

public string GetPassword() => authorizationField.GetPassword();

public void ClearAuthorizationField()
{
    authorizationField.Clear();
}

public void ClearMainField()
{
    outputField.Clear();
    inputField.Clear();
}

public event EventHandler EnterDown;

public event EventHandler CtrlDDown;

private void ViewModel_KeyDown(object sender, KeyEventArgs e)
{
    if(e.KeyCode == Keys.Enter)
    {
        EnterDown?.Invoke(this, EventArgs.Empty);
    }
    if(e.KeyCode == Keys.D && e.Control)
    {
        CtrlDDown?.Invoke(this, EventArgs.Empty);
    }
}
}

```

Чтобы закрыть возможность изменять какие-либо данные внутри самой View, были созданы пользовательские элементы управления.

Authorization field поле для ввода “логина” и “пароля”.

```
public partial class AutorizationField : UserControl
{
    public AutorizationField()
    {
        InitializeComponent();
    }
    public void Clear()
    {
        loginTextBox.Text = String.Empty;
        passwordTextBox.Text = String.Empty;
    }
    public string GetLogin() => loginTextBox.Text;
    public string GetPassword() => passwordTextBox.Text;
}
```

Input field поле для ввода команды.

```
public partial class InputField : UserControl
{
    public InputField()
    {
        InitializeComponent();
    }

    public void Clear()
    {
        textBox.Text = String.Empty;
    }

    public string GetContent()
    {
        return textBox.Text;
    }
}
```

Output field поле для вывода результата выполнения команды

```
public partial class OutputField : UserControl
{
    public OutputField()
    {
        InitializeComponent();
    }
    public void Clear()
    {
        textBox.Text = String.Empty;
    }
    public void PrintText(string content)
    {
        textBox.AppendText(content + Environment.NewLine);
    }
}
```

Status просто отображает текущую директорию.












```
public partial class Status : UserControl
{
    public Status()
    {
        InitializeComponent();
    }

    public string Get()
    {
        return label.Text;
    }

    public void Change(string status)
    {
        label.Text = "[" + status + "] > ";
    }
}
```

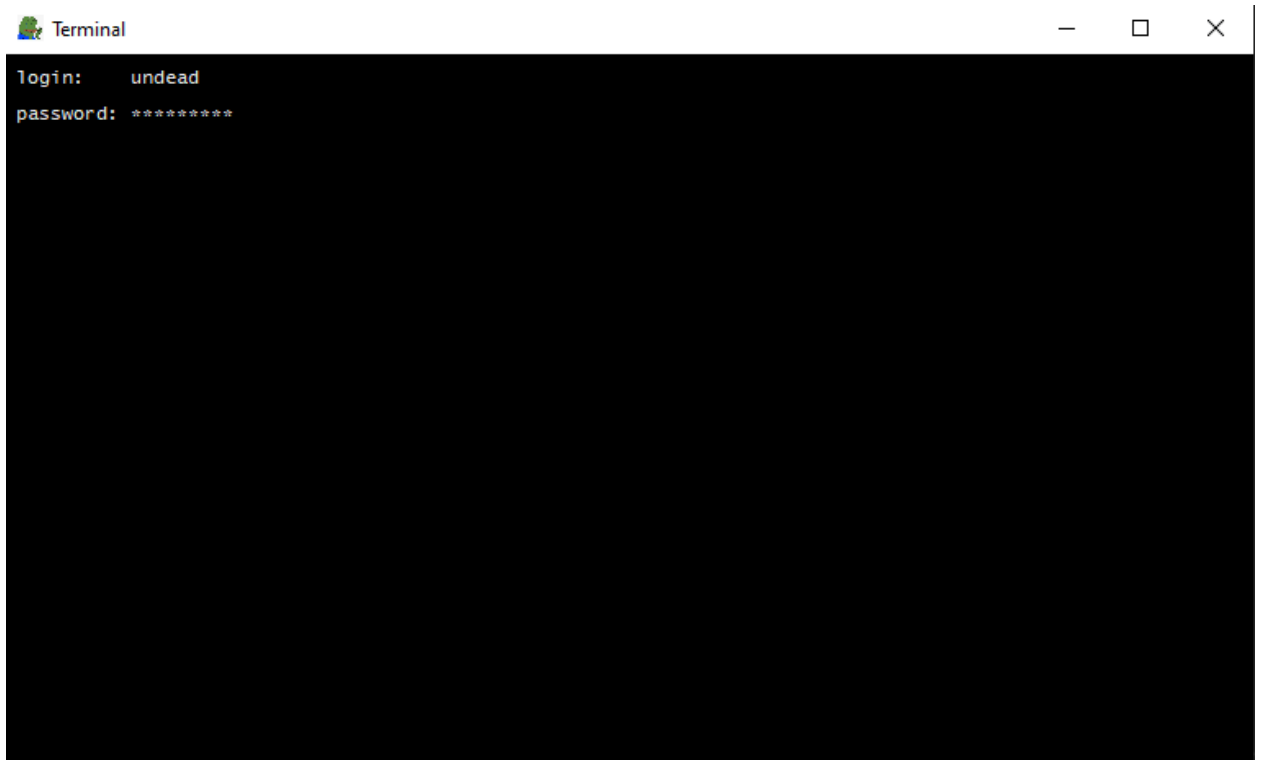
Тестирование программы

Для тестирования используем следующую иерархию каталогов:

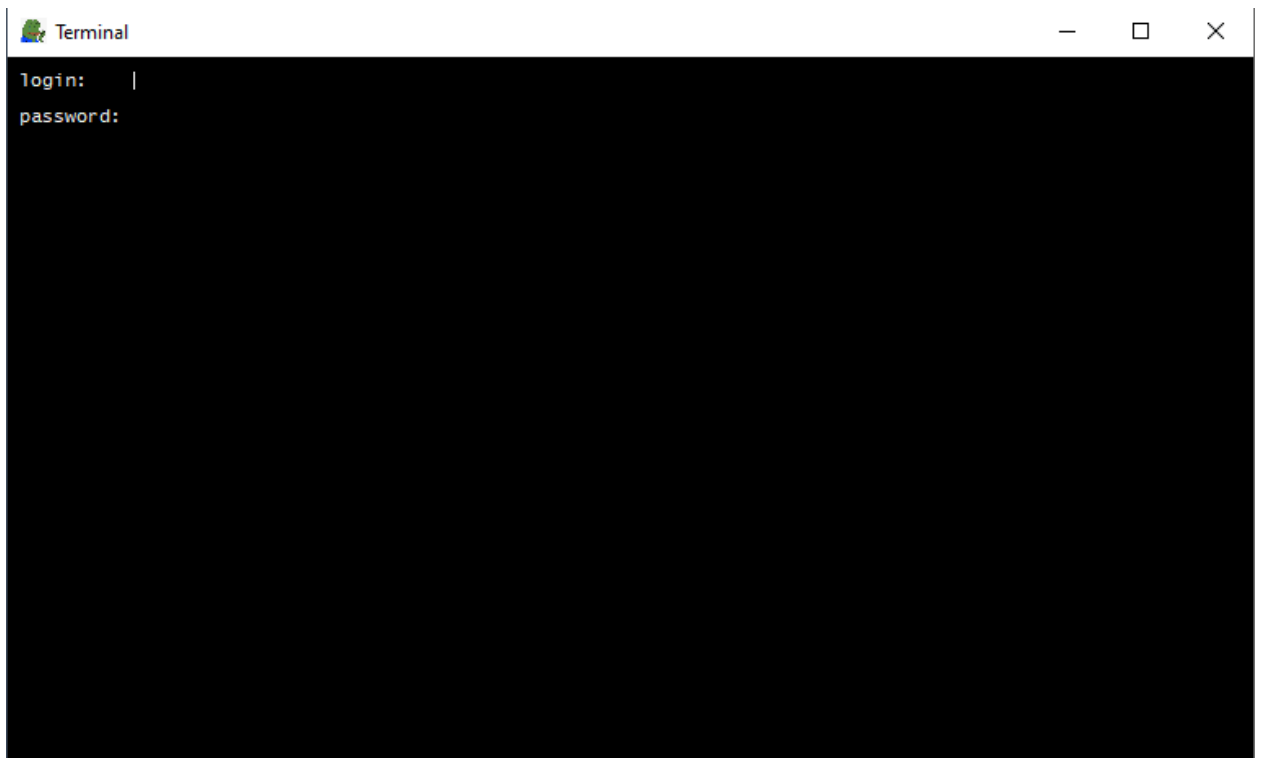
- ▼ ☐  testrgz
 - ▼ ☐  catalog1
 - > ☐  catalog11
 -  (file1.txt)
 - ▼ ☐  catalog2
 -  (file1.txt, file2.txt, file3.txt)
 - ▼ ☐  catalog3
 - > ☐  catalog31
 -  (file2.txt, file3.txt)
 - ☐  da
 -  (abc.txt, bac.txt, file1.txt, file2.txt, file3.txt)

- **Протестируем систему авторизации**

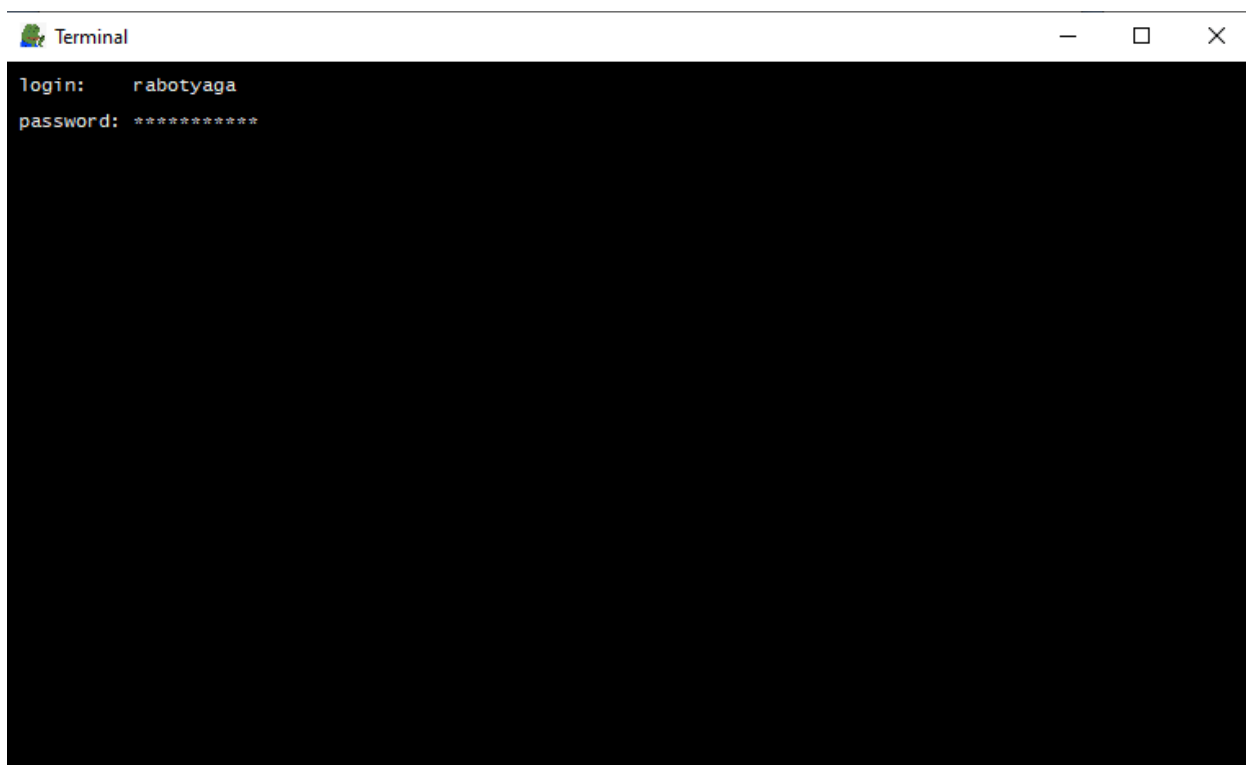
Для начала введем данные несуществующего пользователя.



В результате войти не получилось, поля очистились от введенных данных.



Теперь введем данные существующего пользователя.

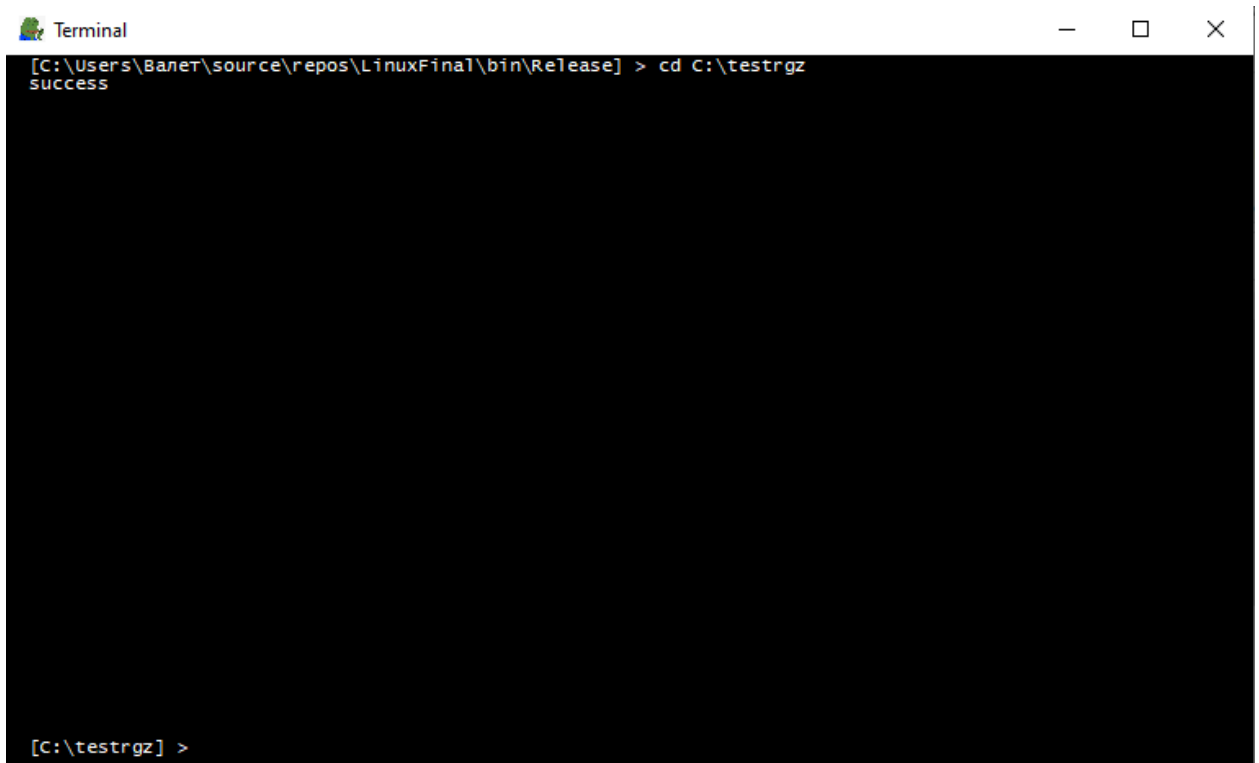


Вход был выполнен.



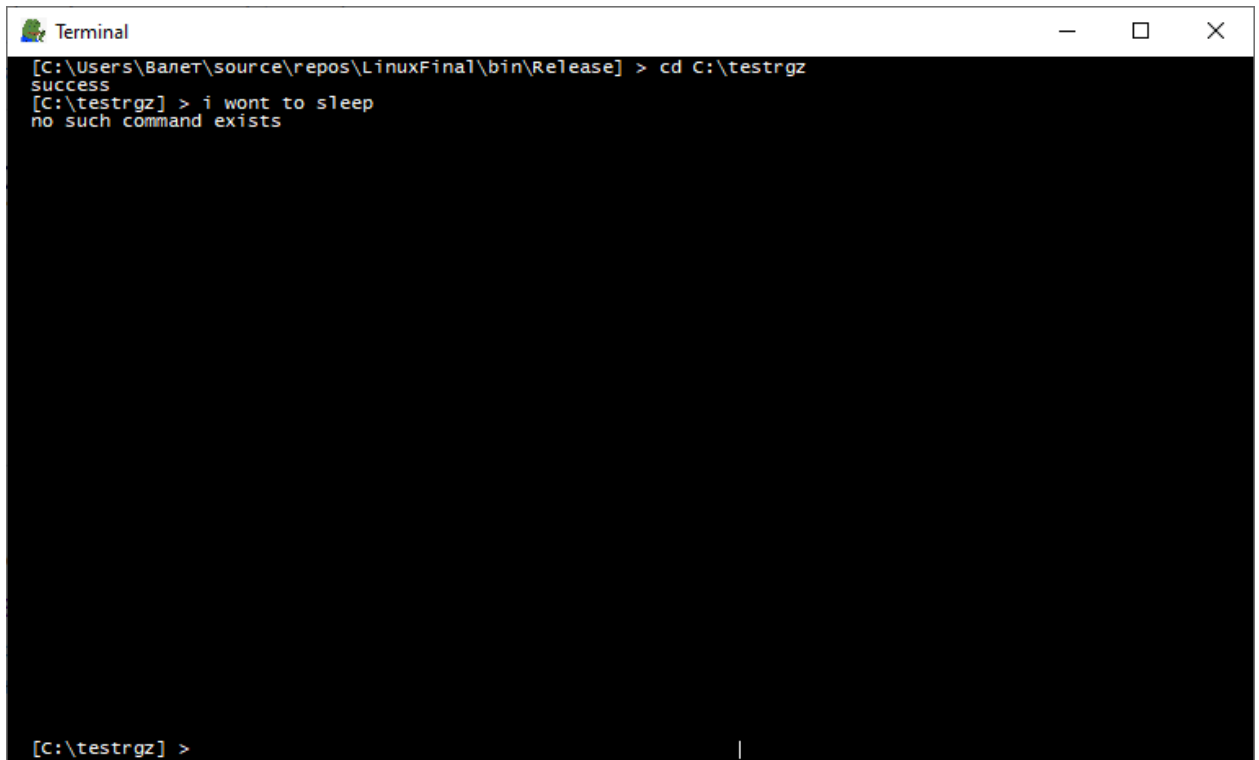
- Протестируем работу в режиме “исполнения команд”

Для начала перейдем в подготовленный каталог, по пути тестируя команду **cd**.

A screenshot of a Windows Terminal window. The title bar shows a green icon and the word "Terminal". The window has standard minimize, maximize, and close buttons. The command prompt shows the current directory as [C:\Users\Baler\source\repos\LinuxFinal\bin\Release]. The user enters the command "cd C:\testrgz", and the prompt returns "success". The prompt then changes to [C:\testrgz] >.

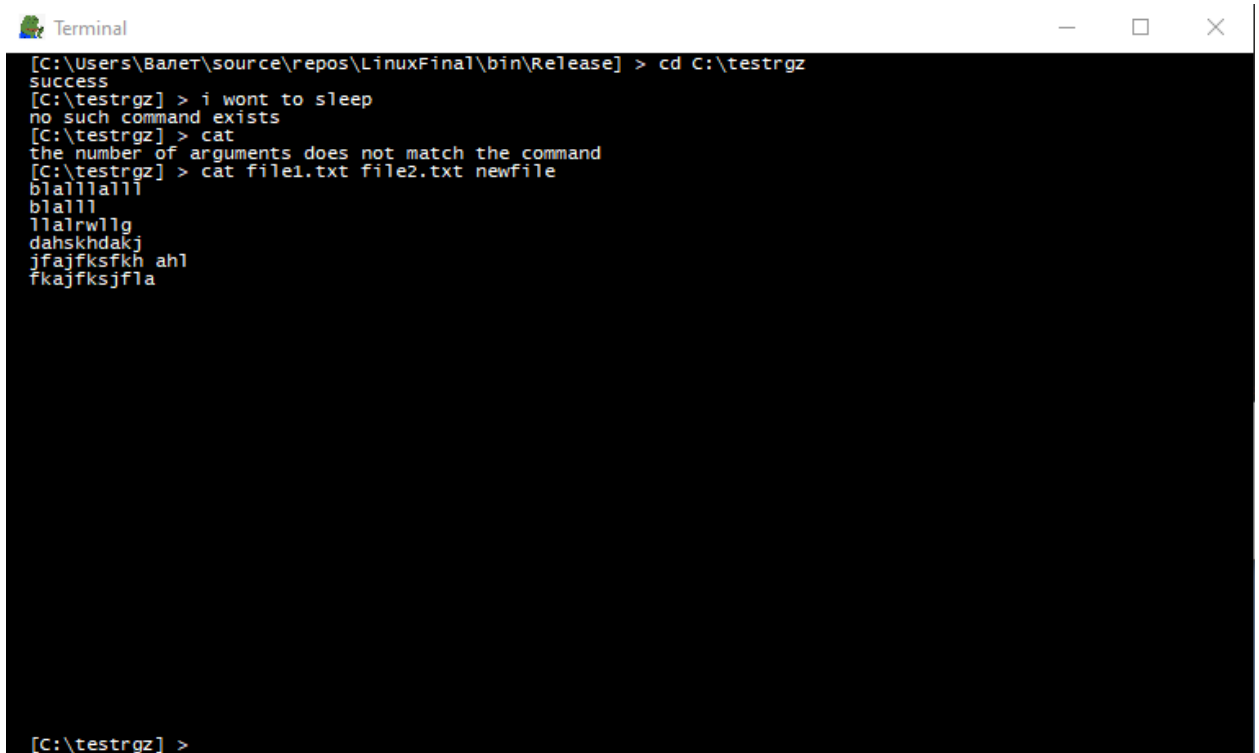
```
Terminal
[C:\Users\Baler\source\repos\LinuxFinal\bin\Release] > cd C:\testrgz
success
[C:\testrgz] >
```

Попробуем ввести несуществующую команду.

A screenshot of a Windows Terminal window, similar to the previous one. The title bar shows a green icon and the word "Terminal". The window has standard minimize, maximize, and close buttons. The command prompt shows the current directory as [C:\Users\Baler\source\repos\LinuxFinal\bin\Release]. The user enters the command "cd C:\testrgz", and the prompt returns "success". The prompt then changes to [C:\testrgz] >. The user enters the command "i wont to sleep", and the prompt returns "no such command exists". The prompt then changes to [C:\testrgz] >.

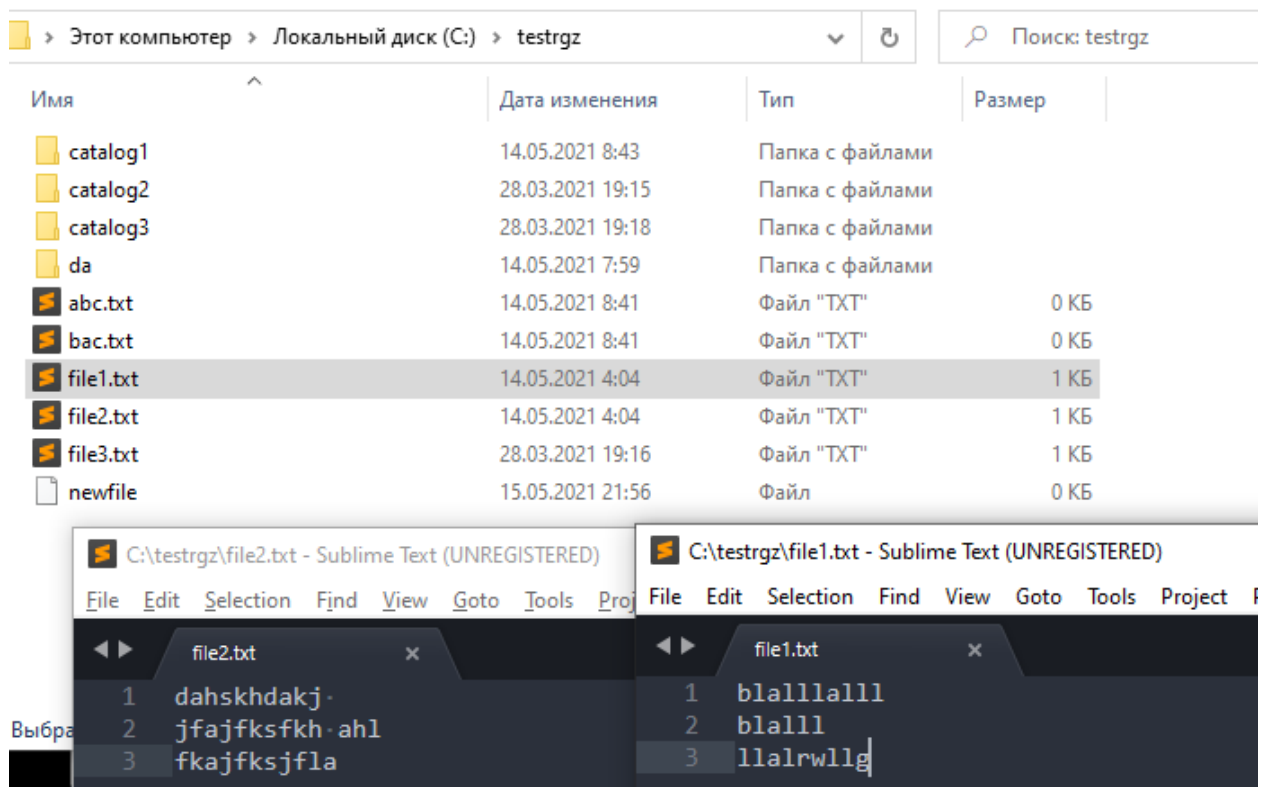
```
Terminal
[C:\Users\Baler\source\repos\LinuxFinal\bin\Release] > cd C:\testrgz
success
[C:\testrgz] > i wont to sleep
no such command exists
[C:\testrgz] >
```

Протестируем команду **cat**



```
Terminal
[C:\Users\Baler\source\repos\LinuxFinal\bin\Release] > cd C:\testrgz
success
[C:\testrgz] > i wont to sleep
no such command exists
[C:\testrgz] > cat
the number of arguments does not match the command
[C:\testrgz] > cat file1.txt file2.txt newfile
blalllalll
blalll
llalrwllg
dahskhdakj
jfajfksfkh ah1
fkajfksjfla
[C:\testrgz] >
```

Так как файла “newfile” не существовало, то он создался после выполнения команды **cat**, также проверим содержимое файлов “file1.txt” и “file2.txt”



Windows Explorer view of the directory C:\testrgz:

Имя	Дата изменения	Тип	Размер
catalog1	14.05.2021 8:43	Папка с файлами	
catalog2	28.03.2021 19:15	Папка с файлами	
catalog3	28.03.2021 19:18	Папка с файлами	
da	14.05.2021 7:59	Папка с файлами	
abc.txt	14.05.2021 8:41	Файл "TXT"	0 КБ
bac.txt	14.05.2021 8:41	Файл "TXT"	0 КБ
file1.txt	14.05.2021 4:04	Файл "TXT"	1 КБ
file2.txt	14.05.2021 4:04	Файл "TXT"	1 КБ
file3.txt	28.03.2021 19:16	Файл "TXT"	1 КБ
newfile	15.05.2021 21:56	Файл	0 КБ

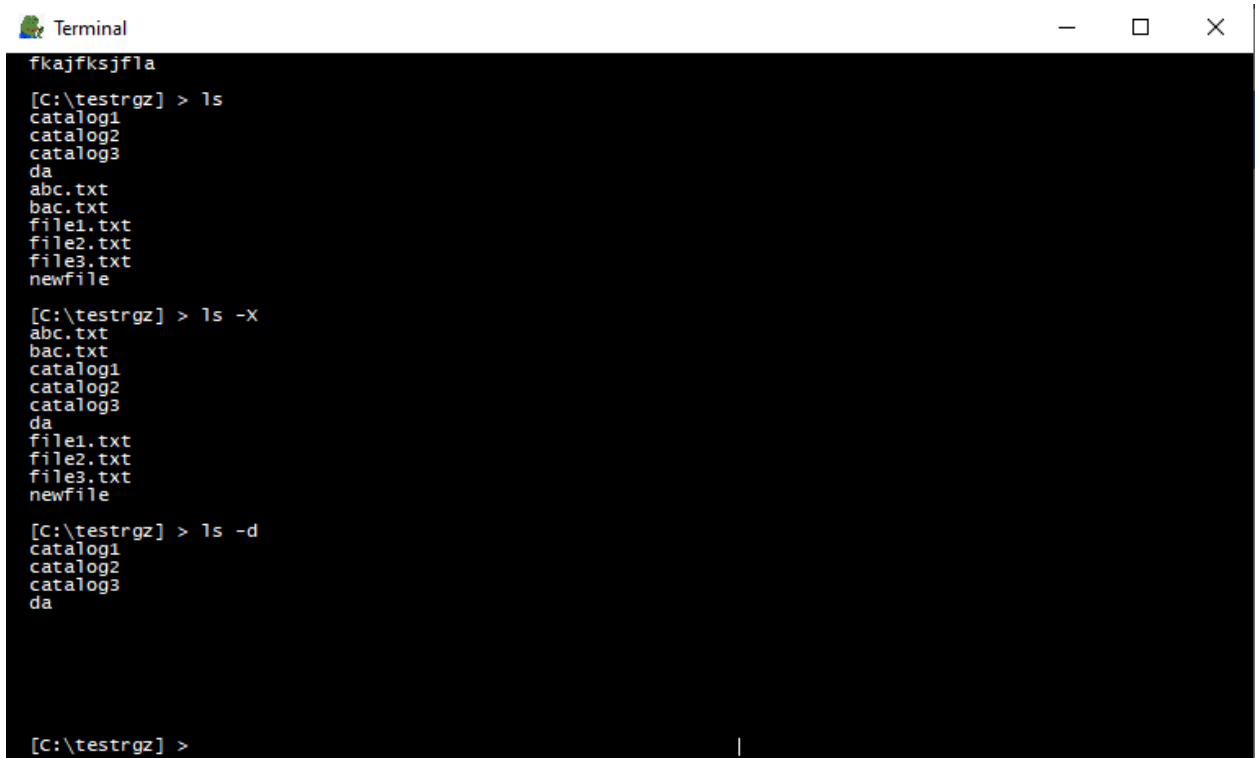
Sublime Text editor windows showing file contents:

- C:\testrgz\file2.txt - Sublime Text (UNREGISTERED)**

```
1 dahskhdakj
2 jfajfksfkh ah1
3 fkajfksjfla
```
- C:\testrgz\file1.txt - Sublime Text (UNREGISTERED)**

```
1 blalllalll
2 blalll
3 llalrwllg
```

Результат исполнения команд `ls`, `ls -X`, `ls -d`



```
fkajfksjfla
[C:\testrgz] > ls
catalog1
catalog2
catalog3
da
abc.txt
bac.txt
file1.txt
file2.txt
file3.txt
newfile

[C:\testrgz] > ls -X
abc.txt
bac.txt
catalog1
catalog2
catalog3
da
file1.txt
file2.txt
file3.txt
newfile

[C:\testrgz] > ls -d
catalog1
catalog2
catalog3
da

[C:\testrgz] >
```

Результат исполнения команды `pwd`



```
fkajfksjfla
[C:\testrgz] > ls
catalog1
catalog2
catalog3
da
abc.txt
bac.txt
file1.txt
file2.txt
file3.txt
newfile

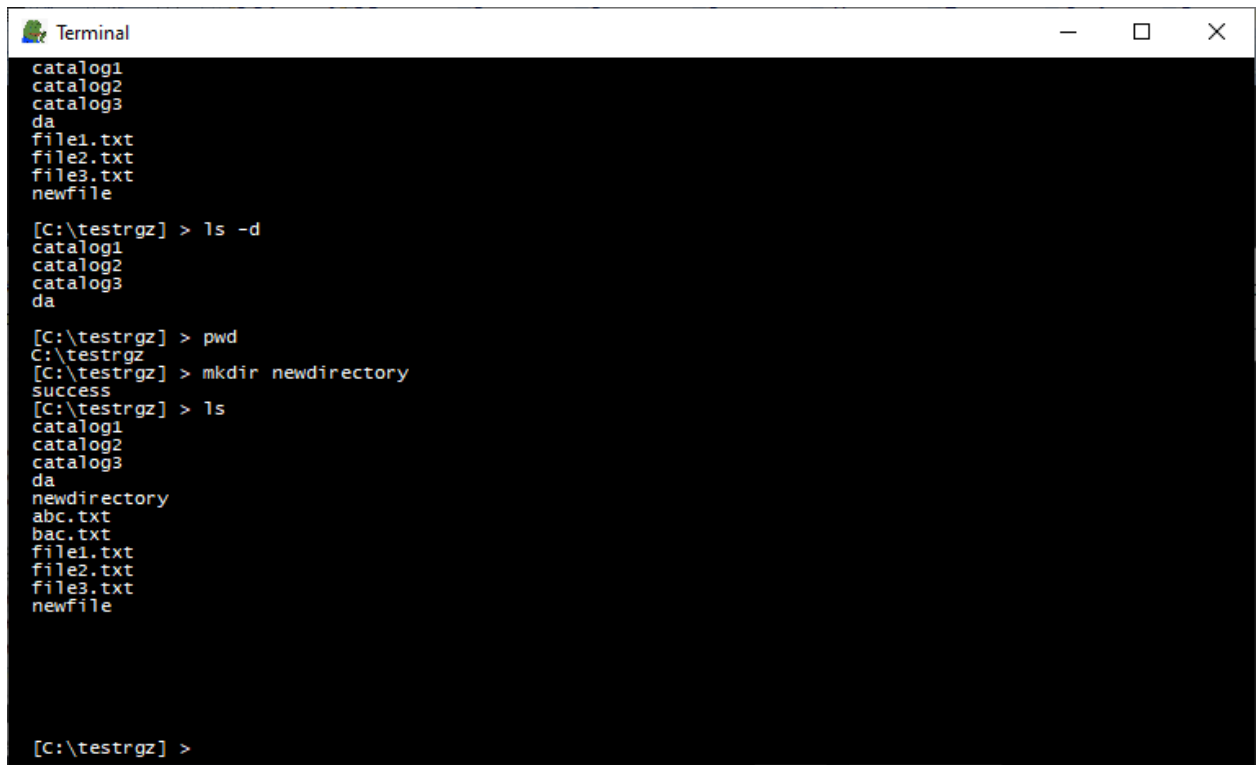
[C:\testrgz] > ls -X
abc.txt
bac.txt
catalog1
catalog2
catalog3
da
file1.txt
file2.txt
file3.txt
newfile

[C:\testrgz] > ls -d
catalog1
catalog2
catalog3
da

[C:\testrgz] > pwd
C:\testrgz

[C:\testrgz] >
```

Создадим новую директорию командой **mkdir** и проверим ее наличие с помощью **ls**

A screenshot of a Windows Terminal window with a black background and white text. The window title is "Terminal". The terminal shows a list of existing files and directories: catalog1, catalog2, catalog3, da, file1.txt, file2.txt, file3.txt, and newfile. Then, the user runs the command "ls -d" which lists the same items. Next, the user runs "pwd" which shows the current directory is "C:\testrgz". Then, the user runs "mkdir newdirectory" and receives a "success" message. Finally, the user runs "ls" which shows the updated list of files and directories, including the newly created "newdirectory" and its contents: abc.txt, bac.txt, file1.txt, file2.txt, file3.txt, and newfile. The prompt "[C:\testrgz] >" is visible at the bottom.

```
catalog1
catalog2
catalog3
da
file1.txt
file2.txt
file3.txt
newfile

[C:\testrgz] > ls -d
catalog1
catalog2
catalog3
da

[C:\testrgz] > pwd
C:\testrgz
[C:\testrgz] > mkdir newdirectory
success
[C:\testrgz] > ls
catalog1
catalog2
catalog3
da
newdirectory
abc.txt
bac.txt
file1.txt
file2.txt
file3.txt
newfile

[C:\testrgz] >
```

Удалим файл с помощью команды **rm**

```
[C:\testrgz] > rm newfile
success
[C:\testrgz] > ls
catalog1
catalog2
catalog3
da
newdirectory
abc.txt
bac.txt
file1.txt
file2.txt
file3.txt

[C:\testrgz] >
```

↑

Этот компьютер

Локальный диск (C:)

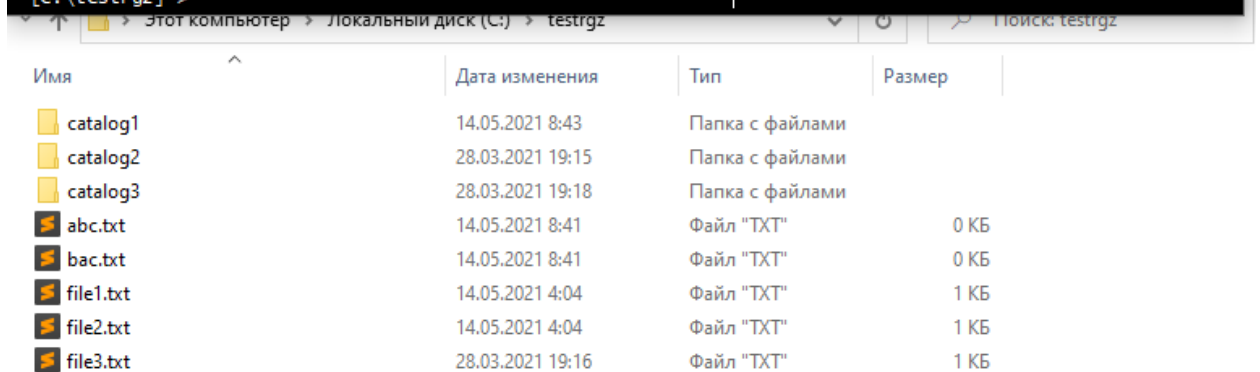
testrgz

Поиск: testrgz

Имя	Дата изменения	Тип	Размер
catalog1	14.05.2021 8:43	Папка с файлами	
catalog2	28.03.2021 19:15	Папка с файлами	
catalog3	28.03.2021 19:18	Папка с файлами	
da	14.05.2021 7:59	Папка с файлами	
newdirectory	15.05.2021 22:08	Папка с файлами	
abc.txt	14.05.2021 8:41	Файл "TXT"	0 КБ
bac.txt	14.05.2021 8:41	Файл "TXT"	0 КБ
file1.txt	14.05.2021 4:04	Файл "TXT"	1 КБ
file2.txt	14.05.2021 4:04	Файл "TXT"	1 КБ
file3.txt	28.03.2021 19:16	Файл "TXT"	1 КБ

Удалим два пустых каталога с помощью команды **rmdir** и **rm -d**

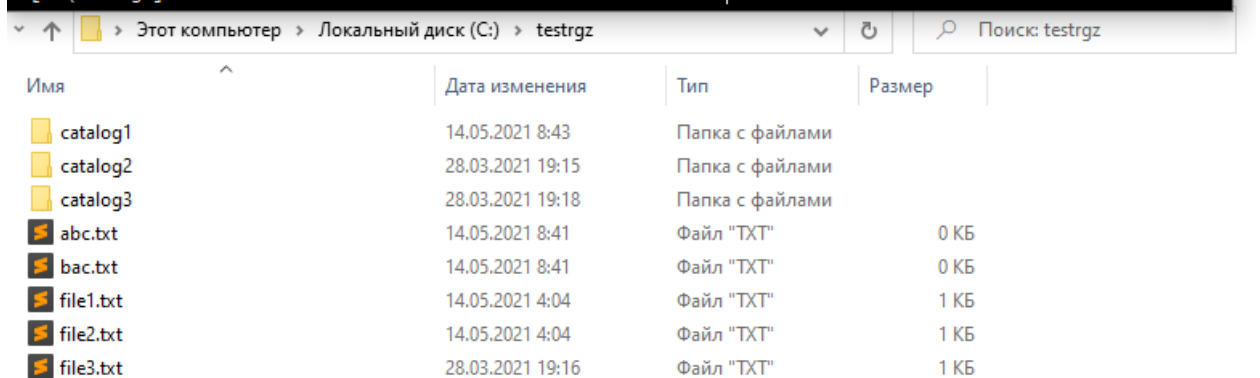
```
[C:\testrgz] > rmdir da
success
[C:\testrgz] > rm -d newdirectory
success
[C:\testrgz] > ls
catalog1
catalog2
catalog3
abc.txt
bac.txt
file1.txt
file2.txt
file3.txt
```



Имя	Дата изменения	Тип	Размер
catalog1	14.05.2021 8:43	Папка с файлами	
catalog2	28.03.2021 19:15	Папка с файлами	
catalog3	28.03.2021 19:18	Папка с файлами	
abc.txt	14.05.2021 8:41	Файл ".TXT"	0 КБ
bac.txt	14.05.2021 8:41	Файл ".TXT"	0 КБ
file1.txt	14.05.2021 4:04	Файл ".TXT"	1 КБ
file2.txt	14.05.2021 4:04	Файл ".TXT"	1 КБ
file3.txt	28.03.2021 19:16	Файл ".TXT"	1 КБ

Проверим работоспособность **date**, **date -u**, **date -r**

```
[C:\testrgz] > date
15.05.2021 22:22:24
[C:\testrgz] > date -u
15.05.2021 15:22:29
[C:\testrgz] > date -r file1.txt
14.05.2021 4:04:47
```



Имя	Дата изменения	Тип	Размер
catalog1	14.05.2021 8:43	Папка с файлами	
catalog2	28.03.2021 19:15	Папка с файлами	
catalog3	28.03.2021 19:18	Папка с файлами	
abc.txt	14.05.2021 8:41	Файл ".TXT"	0 КБ
bac.txt	14.05.2021 8:41	Файл ".TXT"	0 КБ
file1.txt	14.05.2021 4:04	Файл ".TXT"	1 КБ
file2.txt	14.05.2021 4:04	Файл ".TXT"	1 КБ
file3.txt	28.03.2021 19:16	Файл ".TXT"	1 КБ

Результат выполнения **uname**, **uname -o**, **uname -m**

```
[C:\testrgz] > uname -o
Microsoft Windows NT 6.2.9200.0
[C:\testrgz] > uname -m
DESKTOP-JEE2130
[C:\testrgz] > uname
Windows version : Microsoft Windows NT 6.2.9200.0
64 Bit operating system : Yes
Computer name : DESKTOP-JEE2130
Processor count : 6
System directory : C:\WINDOWS\system32
Logical drives : C:
```

```
[C:\testrgz] > |
```


- Некорректно введенные команды

cd несуществующий путь

```
[C:\testrgz] > cd path  
there is no such path
```

cat без аргументов

```
[C:\testrgz] > cat  
the number of arguments does not match the command
```

ls с аргументами

```
[C:\testrgz] > ls arg  
the number of arguments does not match the command
```

rm несуществующий файл

```
[C:\testrgz] > ls  
catalog1  
catalog2  
catalog3  
abc.txt  
bac.txt  
file1.txt  
file2.txt  
file3.txt  
  
[C:\testrgz] > rm f  
there is no such path
```

rmdir несуществующий каталог

```
[C:\testrgz] > ls  
catalog1  
catalog2  
catalog3  
abc.txt  
bac.txt  
file1.txt  
file2.txt  
file3.txt  
  
[C:\testrgz] > rmdir f  
there is no such path
```

rmdir попытка удалить не пустой каталог

```
[C:\testrgz] > rmdir ../  
directory contains files
```

Код программы

Programm.cs

```
static class Program
{
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);

        View view = new View();
        Shell shell = new Shell();
        Terminal terminal = new Terminal(view, shell);
        Application.Run(view);
    }
}
```

Terminal.cs

```
public class Terminal
{
    private IView _view;
    private IShell _shell;
    private IAuthorizationSystem _authorization;
    private ITextHandler<Command> _commandTextHandler;

    public Terminal(IView view, IShell shell)
    {
        _view = view;
        _shell = shell;
        _authorization = new AuthorizationSystem();
        _commandTextHandler = new CommandTextHandler();
        Logout(_view, EventArgs.Empty);
    }

    private void HandleCommand(object sender, EventArgs e)
    {
        string content = _view.GetContentFromCommandLine();
        Command command = _commandTextHandler.Convert(content);

        _view.PrintContent(_view.GetStatus() + content);
        _view.PrintContent(_shell.ExecuteCommand(command.Name,
command.Arguments));
        _view.ChangeStatus(Directory.GetCurrentDirectory());
    }

    private void TryLogin(object sender, EventArgs e)
    {
        User user = new User(_view.GetLogin(), _view.GetPassword());

        if (_authorization.IsExistUser(user.Name, user.Password))
        {
            _view.EnterDown -= TryLogin;
            _view.EnterDown += HandleCommand;
            _view.CtrlDDown += Logout;
            _view.ChangeStatus(Directory.GetCurrentDirectory());
            _view.ShowMainField();
        }
        _view.ClearAuthorizationField();
    }

    private void Logout(object sender, EventArgs e)
    {
        _view.ClearMainField();
        _view.EnterDown -= HandleCommand;
        _view.CtrlDDown -= Logout;
        _view.EnterDown += TryLogin;
        _view.ShowAuthorizationField();
    }
}
```

TextHandler.cs

```
public struct User
{
    public string Name;
    public string Password;
    public User(string name, string password)
    {
        Name = name;
        Password = password;
    }
}

public struct Command
{
    public string Name;
    public string[] Arguments;
    public Command(string name, string[] arguments)
    {
        Name = name;
        Arguments = arguments;
    }
}

public interface ITextHandler<T> where T : struct
{
    T Convert(string text);
}

public class CommandTextHandler : ITextHandler<Command>
{
    public Command Convert(string text)
    {
        string[] words = text.Split();
        if (words.Length > 1)
        {
            return new Command(words[0], words.Skip(1).ToArray());
        }
        return new Command(words[0], null);
    }
}
```

IDataBase.cs

```
public interface IDatabase<T> where T : class
{
    IReadOnlyList<T> GetData();
}
```

AuthorizationSystem.cs

```
public interface IAuthorizationSystem
{
    bool IsExistUser(string userName, string password);
}
public class AuthorizationSystem : IAuthorizationSystem
{
    private IDatabase<IUser> _userDataBase;
    public AuthorizationSystem()
    {
        _userDataBase = new UserDataBase();
    }
    public bool IsExistUser(string userName, string password)
    {
        var users = _userDataBase.GetData();
        foreach(IUser user in users)
        {
            if(userName == user.Name && password == user.Password)
            {
                return true;
            }
        }
        return false;
    }
}
```

User.cs

```
public interface IUser
{
    string Name { get; }
    string Password { get; }
}
public class Root : IUser
{
    public string Name => "root";
    public string Password => "admin1234";
}

public class Hunky : IUser
{
    public string Name => "rabotyaga";
    public string Password => "15kpermonth";
}
```

UserDataBase.cs

```
public class UserDataBase : IDatabase<IUser>
{
    private List<IUser> _userList;
    public UserDataBase()
    {
        _userList = new List<IUser>();
        _userList.Add(new Root());
        _userList.Add(new Hunky());
    }
    public IReadOnlyList<IUser> GetData()
    {
        return _userList.AsReadOnly();
    }
}
```

Shell.cs

```
public interface IShell
{
    string ExecuteCommand(string commandName, string[] arguments);
}

public class Shell : IShell
{
    private IDatabase<ICommand> _commandDataBase;
    public Shell()
    {
        _commandDataBase = new CommandDataBase();
    }

    public string ExecuteCommand(string commandName, string[] arguments)
    {
        var commands = _commandDataBase.GetData();
        foreach (ICommand command in commands)
        {
            if (command.Name == commandName)
            {
                return command.Execute(arguments);
            }
        }
        return NonExistentCommand.Execute();
    }
}
```

ICommand.cs

```
public interface ICommand
{
    string Name { get; }
    string Execute(string[] arguments);
}
```

CommandDataBase.cs

```
public class CommandDataBase : IDatabase<ICommand>
{
    private List<ICommand> _commandList;
    public CommandDataBase()
    {
        _commandList = new List<ICommand>();
        _commandList.Add(new PrintWorkingDirectory());
        _commandList.Add(new FileList());
        _commandList.Add(new ChangeDirectory());
        _commandList.Add(new Concatenate());
        _commandList.Add(new MakeDirectory());
        _commandList.Add(new UnixName());
        _commandList.Add(new RemoveDirectory());
        _commandList.Add(new Remove());
        _commandList.Add(new FileType());
        _commandList.Add(new ShowDate());
        //Add more commands
    }

    public IReadOnlyList<ICommand> GetData()
    {
        return _commandList.AsReadOnly();
    }
}
```

NonExistentCommand.cs

```
public static class NonExistentCommand
{
    public static string Execute()
    {
        return "no such command exists";
    }
}
```

ChangeDirectory.cs

```
public class ChangeDirectory : ICommand
{
    public string Name => "cd";

    public string Execute(string[] arguments)
    {
        if (arguments == null)
        {
            if (Directory.Exists("C:\\"))
            {
                Directory.SetCurrentDirectory("C:\\");
                return "success";
            }
        }
        if (arguments.Length == 1)
        {
            if (Directory.Exists(arguments[0]))
            {
                Directory.SetCurrentDirectory(arguments[0]);
                return "success";
            }
            return "there is no such path";
        }
        return "the number of arguments does not match the command";
    }
}
```

Concatenate.cs

```
public class Concatenate : ICommand
{
    public string Name => "cat";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            StringBuilder stringBuilder = new StringBuilder();

            foreach (string argument in arguments)
            {
                if (File.Exists(argument))
                {
                    stringBuilder.AppendLine(File.ReadAllText(argument));
                }
                else
                {
                    File.Create(argument).Close();
                }
            }
            return stringBuilder.ToString();
        }
        return "the number of arguments does not match the command";
    }
}
```


FileList.cs

```
public class FileList : ICommand
{
    public string Name => "ls";

    public string Execute(string[] arguments)
    {
        var output = new StringBuilder();
        var directoryContent = new List<string>();

        if (arguments != null)
        {
            if (arguments[0] == "-d")
            {
                directoryContent =
Directory.GetDirectories(Directory.GetCurrentDirectory()).ToList();
            }
            else if (arguments[0] == "-X")
            {
                directoryContent =
Directory.GetDirectories(Directory.GetCurrentDirectory()).ToList();

                directoryContent.AddRange(Directory.GetFiles(Directory.GetCurrentDirectory())
.ToList());

                directoryContent.Sort();
            }
            else
            {
                return "the number of arguments does not match the
command";
            }
        }
        else
        {
            directoryContent =
Directory.GetDirectories(Directory.GetCurrentDirectory()).ToList();

            directoryContent.AddRange(Directory.GetFiles(Directory.GetCurrentDirectory())
.ToList());
        }

        foreach (var content in directoryContent)
        {
            output.AppendLine(content.Split('\\').Last());
        }

        return output.ToString();
    }
}
```

FileType.cs

```
public class FileType : ICommand
{
    public string Name => "file";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            if (arguments.Length == 1)
            {
                return Path.GetExtension(Directory.GetCurrentDirectory()
+ arguments[0]);
            }
            return "the number of arguments does not match the command";
        }
    }
}
```

MakeDirectory.cs

```
public class MakeDirectory : ICommand
{
    public string Name => "mkdir";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            if (arguments.Length == 1)
            {
                if (!Directory.Exists(arguments[0]))
                {
                    Directory.CreateDirectory(arguments[0]);
                    return "success";
                }
                return "directory already exists";
            }
        }
        return "the number of arguments does not match the command";
    }
}
```

PrintWorkingDirectory.cs

```
public class PrintWorkingDirectory : ICommand
{
    public string Name => "pwd";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            return "the number of arguments does not match the command";
        }
        return Directory.GetCurrentDirectory();
    }
}
```

RemoveDirectory.cs

```
public class RemoveDirectory : ICommand
{
    public string Name => "rmdir";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            if (arguments.Length == 1)
            {
                if (Directory.Exists(arguments[0]))
                {
                    if (!Directory.GetFiles(arguments[0]).Any() &&
!Directory.GetDirectories(arguments[0]).Any())
                    {
                        Directory.Delete(arguments[0]);
                        return "success";
                    }
                    return "directory contains files";
                }
                return "there is no such path";
            }
        }
        return "the number of arguments does not match the command";
    }
}
```

Remove.cs

```
public class Remove : ICommand
{
    public string Name => "rm";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            if (arguments[0] == "-d")
            {
                if (Directory.Exists(arguments[1]))
                {
                    if (!Directory.GetFiles(arguments[1]).Any() &&
!Directory.GetDirectories(arguments[1]).Any())
                    {
                        Directory.Delete(arguments[1]);
                        return "success";
                    }
                    return "there is no such path";
                }
                return "directory contains files";
            }
            else
            {
                if (arguments.Length == 1)
                {
                    if (File.Exists(arguments[0]))
                    {
                        File.Delete(arguments[0]);
                        return "success";
                    }
                    return "there is no such path";
                }
            }
        }
        return "the number of arguments does not match the command";
    }
}
```

ShowDate.cs

```
public class ShowDate : ICommand
{
    public string Name => "date";

    public string Execute(string[] arguments)
    {
        if (arguments != null)
        {
            if (arguments[0] == "-r" && arguments.Length == 2)
            {
                if (File.Exists(arguments[1]))
                {
                    return
File.GetLastWriteTime(arguments[1]).ToString();
                }
                return "file does not exist";
            }
            if (arguments[0] == "-u")
            {
                return DateTime.UtcNow.ToString();
            }
        }
        return DateTime.Now.ToString();
    }
}
```

UnixName.cs

```
public static class BasicSystemInfo
{
    public static string ToString()
    {
        StringBuilder output = new StringBuilder();
        output.AppendFormat("Windows version : {0}\r\n",
Environment.OSVersion);
        output.AppendFormat("64 Bit operating system : {0} \r\n",
Environment.Is64BitOperatingSystem? "Yes" : "No");
        output.AppendFormat("Computer name : {0} \r\n",
Environment.MachineName);
        output.AppendFormat("Processor count : {0} \r\n",
Environment.ProcessorCount);
        output.AppendFormat("System directory : {0} \r\n",
Environment.SystemDirectory);
        output.AppendFormat("Logical drives : {0} \r\n",
            String.Join(", ", Environment.GetLogicalDrives())
                .Replace("\\", String.Empty));
        return output.ToString();
    }
}

class UnixName : ICommand
{
    public string Name => "uname";

    public string Execute(string[] arguments)
    {
        if(arguments == null)
        {
            return BasicSystemInfo.ToString();
        }
        else
        {
            if (arguments[0] == "-o")
            {
                return Environment.OSVersion.ToString();
            }
            if(arguments[0] == "-m")
            {
                return Environment.MachineName;
            }
        }
        return "the number of arguments does not match the command";
    }
}
```

View.cs

```
public interface IView
{
    void ClearAuthorizationField();
    void ClearMainField();
    void ShowAuthorizationField();
    void ShowMainField();
    void PrintContent(string content);
    string GetContentFromCommandLine();
    string GetLogin();
    string GetPassword();
    void ChangeStatus(string status);
    string GetStatus();

    event EventHandler EnterDown;
    event EventHandler CtrlDDown;
}

public partial class View : Form, IView
{
    public View()
    {
        InitializeComponent();
    }

    public string GetContentFromCommandLine()
    {
        var content = inputField.GetContent();
        inputField.Clear();
        return content;
    }

    public void PrintContent(string content)
    {
        outputField.PrintText(content);
    }

    public void ChangeStatus(string status)
    {
        this.status.Change(status);
    }

    public string GetStatus()
    {
        return status.Get();
    }

    public void ShowAuthorizationField()
    {
        authorizationField.Visible = true;
        status.Visible = false;
        inputField.Visible = false;
        outputField.Visible = false;
    }

    public void ShowMainField()
    {
        authorizationField.Visible = false;
        status.Visible = true;
        inputField.Visible = true;
        outputField.Visible = true;
        inputField.Select();
    }

    public string GetLogin() => authorizationField.GetLogin();
    public string GetPassword() => authorizationField.GetPassword();
    public void ClearAuthorizationField()
    {
        authorizationField.Clear();
    }
}
```

```
}  
public void ClearMainField()  
{  
    outputField.Clear();  
    inputField.Clear();  
}  
  
public event EventHandler EnterDown;  
public event EventHandler CtrlDDown;  
  
private void ViewModel_KeyDown(object sender, KeyEventArgs e)  
{  
    if(e.KeyCode == Keys.Enter)  
    {  
        EnterDown?.Invoke(this, EventArgs.Empty);  
    }  
    if(e.KeyCode == Keys.D && e.Control)  
    {  
        CtrlDDown?.Invoke(this, EventArgs.Empty);  
    }  
}  
}
```


Status.cs

```
public partial class Status : UserControl
{
    public Status()
    {
        InitializeComponent();
    }

    public string Get()
    {
        return label.Text;
    }

    public void Change(string status)
    {
        label.Text = "[" + status + "] > ";
    }
}
```

OutputField.cs

```
public partial class OutputField : UserControl
{
    public OutputField()
    {
        InitializeComponent();
    }

    public void Clear()
    {
        textBox.Text = String.Empty;
    }

    public void PrintText(string content)
    {
        textBox.AppendText(content + Environment.NewLine);
    }
}
```

InputField.cs

```
public partial class InputField : UserControl
{
    public InputField()
    {
        InitializeComponent();
    }

    public void Clear()
    {
        textBox.Text = String.Empty;
    }

    public string GetContent()
    {
        return textBox.Text;
    }
}
```

AuthorizationField.cs

```
public partial class AuthorizationField : UserControl
{
    public AuthorizationField()
    {
        InitializeComponent();
    }
    public void Clear()
    {
        loginTextBox.Text = String.Empty;
        passwordTextBox.Text = String.Empty;
    }
    public string GetLogin() => loginTextBox.Text;
    public string GetPassword() => passwordTextBox.Text;
}
```

~~*надеюсь ничего не забыл.~~