**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**

**МЕХАНИКИ И ОПТИКИ»**

**Факультет «ИКТ»**

**Дисциплина**

«Создание программного обеспечения инфокоммуникационных систем»

**Лабораторная работа 2**

**«**Разработка структурной модели – диаграммы классов**»**

**Выполнил:**

студент группы K3415

Лобченко Д. Р.

**Проверил**:

Осипов Н. А.

Санкт-Петербург

2021

**Практическое задание 4. Разработка пользовательского интерфейса**

**Упражнение 1. Использование простых элементов управления**

1. Создадим новый проект WPF Application с именем «UserIn1».
2. Определим пользовательский интерфейс приложения.
3. Определим свойства добавленных элементов в соответствии с заданием, чтобы связать элементы Label с элементами ввода текста.

Привязка позволяет производить навигацию по элементам формы с помощью комбинации клавиш Alt + клавиша быстрого доступа. После ввода комбинации связанное поле textbox оказывается выбранным.

1. Построим и протестируем приложение.

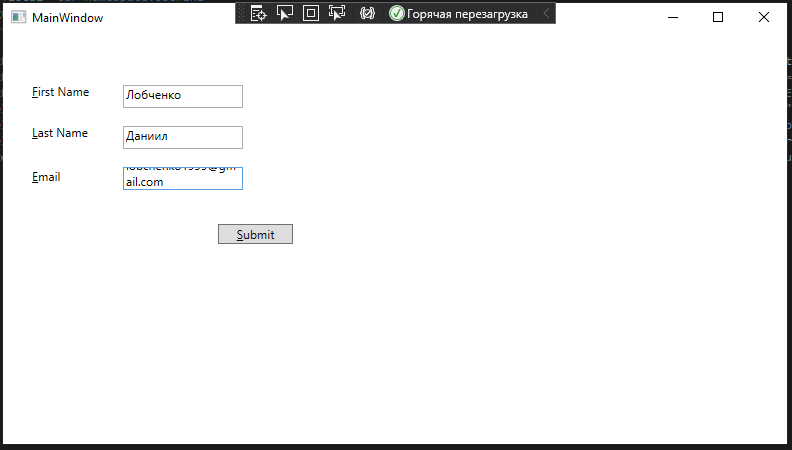


Рисунок 1 - Тестирование приложения

**Практическое задание 4. Разработка пользовательского интерфейса**

**Упражнение 2. Размещение связанных элементов**

1. Создадим новый проект WPF Application с именем «UserIn2».
2. Определим пользовательский интерфейс приложения.
3. Определим свойства добавленных элементов в соответствии с заданием, включая 4 элемента в панели инструментов.

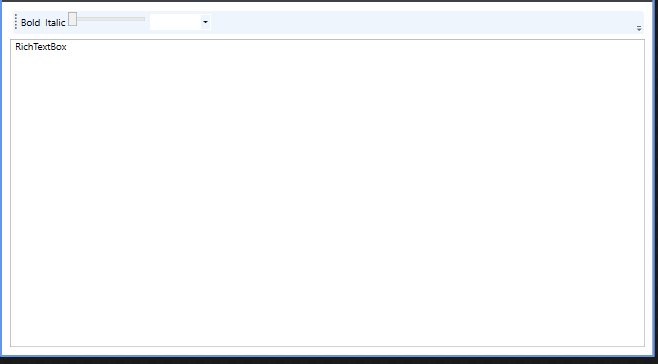


Рисунок 2 - Пользовательский интерфейс приложения

1. Добавим обработчики события **Click** для элементов кнопок на панели инструментов. Для элемента Slider добавим обработчик события **SliderChanged**, а для элемента ComboBox события **ComboBoxSelectionChanged**.
2. Добавим инициализацию элемента ComboBox коллекцией стандартных шрифтов в конструкторе формы.
3. Запустим и протестируем приложение.

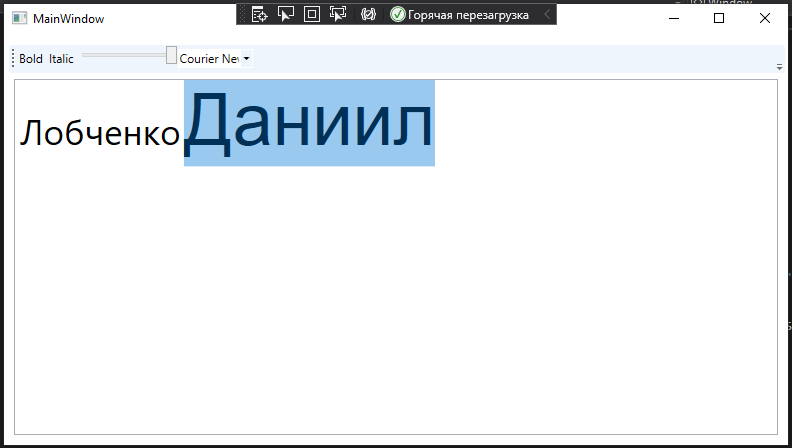


Рисунок 3 - Тестирование приложения.

Как и ожидалось элементы меню меняют оформление выделенного текста.

**Практическое задание 4. Разработка пользовательского интерфейса**

**Упражнение 3. Работа с элементами управления раскладкой**

1. Создадим новый проект WPF Application с именем «UserIn3».
2. В представлении XAML изменим открывающий и закрывающий теги **Grid** тегамии **DockPanel**.
3. Внесем изменения в представление XAML формы в соответствии с условиями задания.

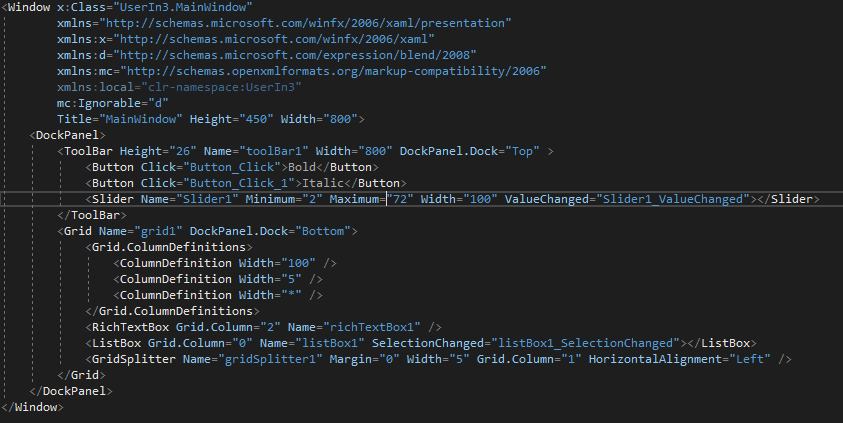


Рисунок 4 - XAML представление формы.

1. Определим обработчики событий для созданных пользовательских элементов.

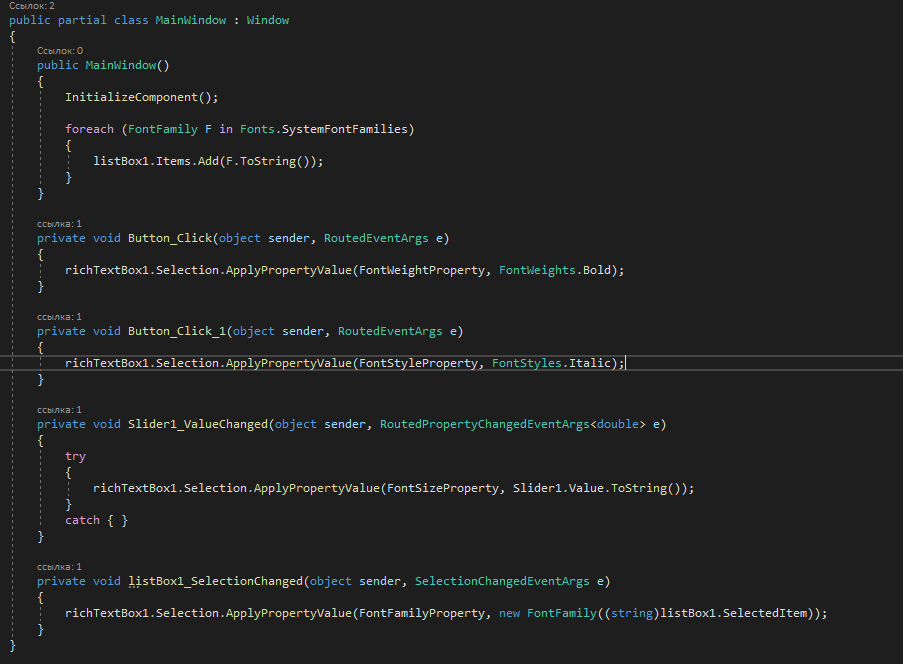


Рисунок 5 - класс формы с определенными обработчиками событий

1. Построим и протестируем приложение.

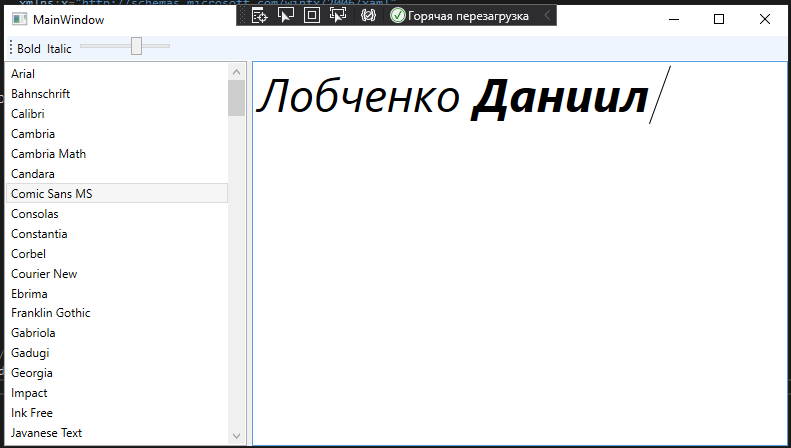


Рисунок 6 - тестирование приложения

**Практическое задание 5. Использование компонента BackgroundWorker**

**Упражнение 1. Работа с компонентом BackgroundWorker**

**Теоретическая вставка**

Класс **BackgroundWorker** появился в .NET 2.0 и был предназначен для упрощения работы с потоками в приложениях Windows Forms. Однако BackgroundWorker в той же мере применим и в WPF. Компонент BackgroundWorker предоставляет почти идеальный способ запуска длительно выполняющихся задач в отдельном потоке. Он использует диспетчер "за кулисами" и абстрагирует сложности маршализации с помощью модели событий.

По умолчанию код, вызываемый приложением, выполняется в основном потоке этого приложения. Таким образом, во время работы этого кода не выполняются никакие другие действия, в том числе и обновление интерфейса.

**BackgroundWorker** незаменим, если есть единственная асинхронная задача, которая выполняется в фоновом режиме от начала до конца (с необязательной поддержкой уведомлений о продвижении и возможностью отмены).

В классе BackgroundWorker реализованы три основных события: **ProgressChanged, RunWorkerCompleted и DoWork. При этом первые два выполняются в основном потоке программы и могут влиять на пользовательский интерфейс, а DoWork выполняется в отдельном потоке.**

**Создание BackgroundWorker**

Чтобы использовать BackgroundWorker, следует начать с создания его экземпляра. При этом на выбор доступны два подхода:

* Можно создать BackgroundWorker в коде и присоединить программно все обработчики событий.
* Можно объявить BackgroundWorker в XAML-разметке. Преимущество такого подхода в возможности присоединения обработчиков событий через атрибуты. Поскольку BackgroundWorker не является видимым элементом WPF, его нельзя поместить куда угодно. Вместо этого его понадобится объявить, как ресурс для окна.

**Выполнение работы**

1. Создадим новый проект WPF Application с именем «Window\_Main».
2. Определим пользовательский интерфейс приложения.
3. Определим свойства добавленных элементов в соответствии с заданием.
4. Создадим новый экземпляр класса BackgroundWorker и определим для него обработчики событий **DoWork** и **RunWorkerCompleted**.
5. Определим обработчики событий **Click** для кнопок **Start** и **Canсel.**

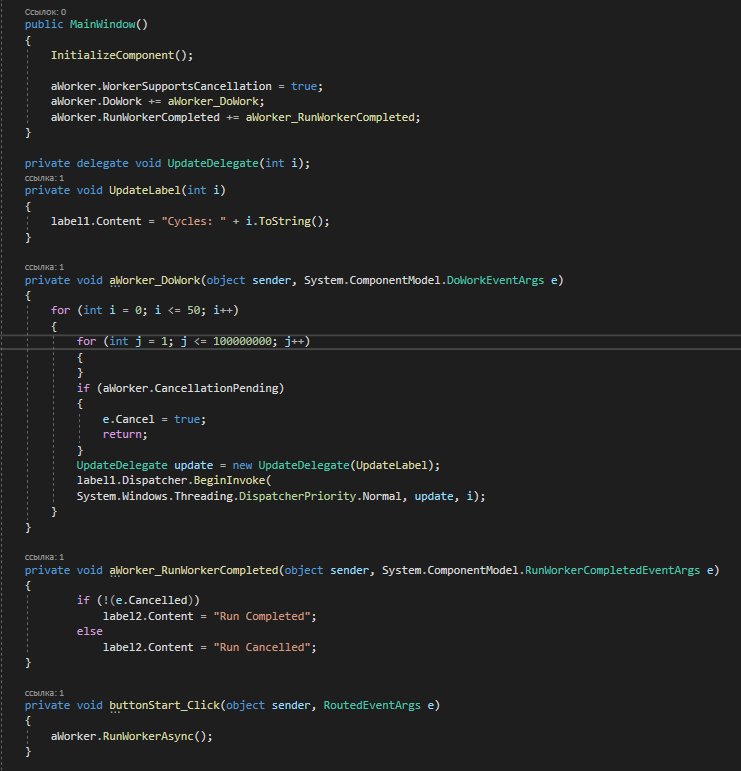


Рисунок 7 - класс MainWindow. Определение компонента BackgroundWorker.

1. Построим и протестируем приложение.

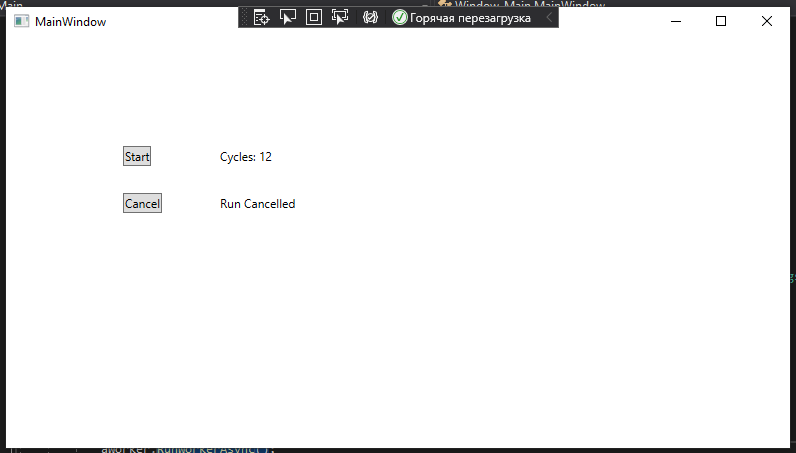


Рисунок 8 - Тестирование приложения.

**Практическое задание 6. Настройка интерфейса пользователя**

**Упражнение 1. Использование элементов Windows Forms**

**Теоретическая вставка**

Для того чтобы упростить совместимость между WPF и WinForms, разработчики Microsoft позаботились о том, чтобы элементы управления WinForms можно было использовать в приложениях WPF. Этот механизм реализован в качестве WindowsFormsHost.

**Выполнение работы**

1. Создадим новый проект WPF Application с именем «UserForms».
2. Определим пользовательский интерфейс приложения.
3. Определим свойства добавленных элементов в соответствии с заданием.
4. В классе формы определим коллекцию номеров с именем **PhoneNumbers** и окно **SaveDialogBox** из Windows Forms.
5. Добавим обработчики события Click для кнопок «**Add To Collection**» и «**Save Collection To File**».
6. Построим и протестируем приложение.

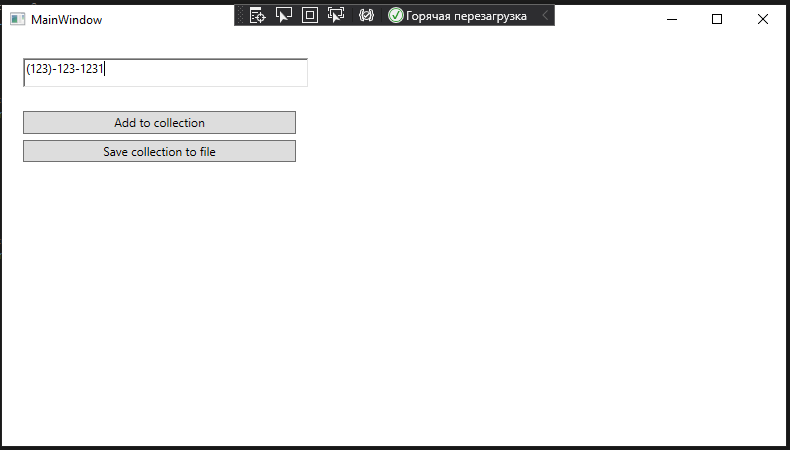


Рисунок 9 - проверка работы приложения

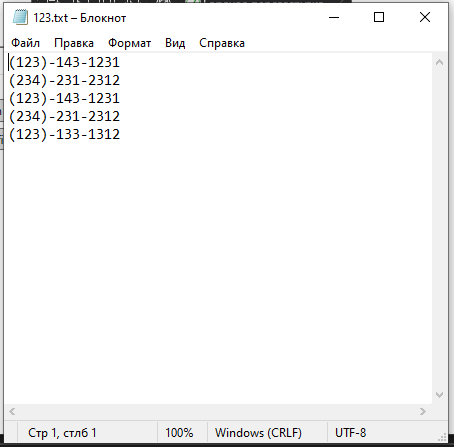


Рисунок 10 - проверка сохранения в файл

**Практическое задание 6. Настройка интерфейса пользователя**

**Упражнение 2. Построение специального свойства зависимости**

1. Создадим новый проект WPF Application с именем «WpfUserControl».
2. Добавим в проект пользовательский элемент управления с именем **ShowNumberControl.**
3. Отредактируем XAML файл, связанный с созданным элементом управления в соответствии с условием задания, добавив в **Grid** элемент управления **Label**.
4. Отредактируем файл кода данного элемента определив параметр **CurrentNumber** встроенным способом.
5. В коде разметки XAML окна **MainWindow** объявим экземпляр специального элемента управления внутри диспетчера компоновки и определим специальное пространство имен XML, отображаемое на данный элемент.
6. Добавим код для применения анимации для компонента.
7. Реализуем статический метод для проверки достоверности данных с именем **ValidateCurrentNumber** и функционалом, описанным в условии задания, и зададим его как финальный аргумент в метод DependencyProperty.Register.
8. Реализуем статический метод для реагирования на изменение свойства с именем **CurrentNumberChanged** и функционалом, описанным в условии задания, и зададим его как второй аргумент в конструктор UIPropertyMetadata.
9. Построим и протестируем приложение.

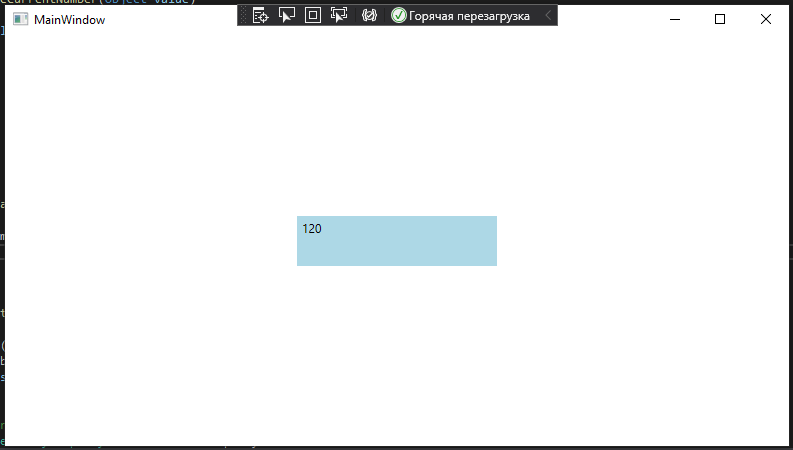


Рисунок 11 - тестирование приложения

Как и ожидалось происходит анимация контента компонента от 100 до 200.

**Практическое задание 7. Графика в приложениях WPF**

**Упражнение 1. Рисование замкнутой фигуры с помощью элемента "Многоугольник"**

1. Создадим новый проект WPF Application с именем «WpfAplicationPolygon».
2. Отредактируем файл MainWindow.xaml в соответствии с условием задания, для отображения четырех треугольников, а также двух звезд.
3. Построим и протестируем приложение.

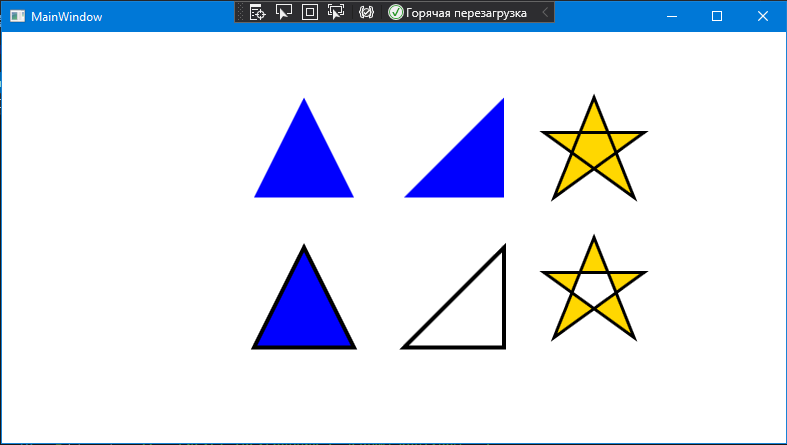


Рисунок 12 - тестирование приложения

**Практическое задание 7. Графика в приложениях WPF**

**Упражнение 2. Создание приложения рисования**

1. Создадим новый проект WPF Application с именем «InkCanvas».
2. Определим пользовательский интерфейс приложения. В том числе добавим элемент **InkCanvas**.
3. Определим свойства добавленных элементов в соответствии с заданием.
4. Создадим обработчики событий Click для элементов Button (кнопки «**Close**» и «**Clear**»).
5. Построим и протестируем приложение.

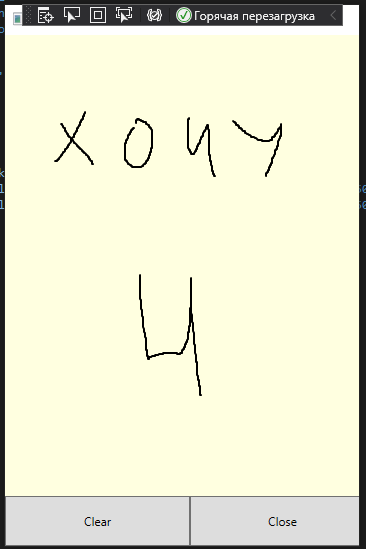


Рисунок 13 - Тестирование приложения

**Практическое задание 7. Графика в приложениях WPF**

**Упражнение 3. Создание диаграмм с использованием WPF Toolkit**

1. Создадим новый проект WPF Application с именем «WpfApplicationChart».
2. Определим зависимости System.Windows.Controls.DataVisualization.Toolkit.dll и WPFToolkit.dll.
3. Импортирует необходимые имена и пространства имен в файле WindowChartSample.xaml.
4. Добавим в элемент Grid диаграмму (элемент **chartingToolkit:Chart**).
5. Определим свойства элементов интерфейса в соответствии с заданием.
6. Добавим в проект класс **Model** и отредактируем в соответствии с заданием, определив класс **Data** и реализовав хранение данных в списке.
7. Добавим секцию **Window.Resources** в файл **Window.xaml** и свяжем их с моделью.
8. Построим и протестируем приложение.

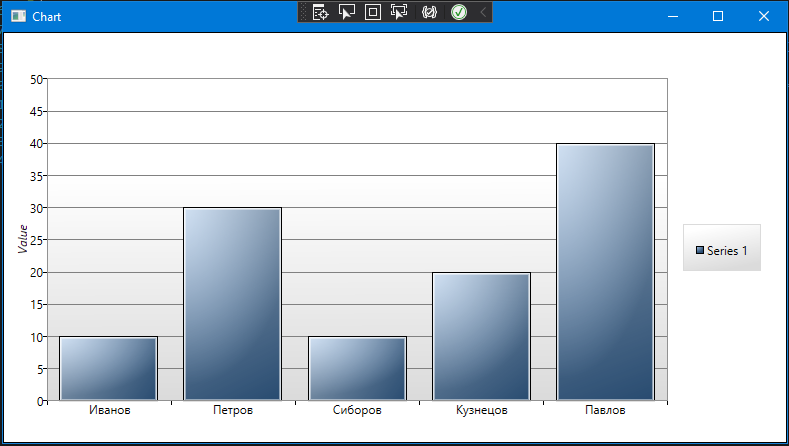


Рисунок 14 - Тестирование приложения

**Вывод**

В данной лабораторной работе были рассмотрены:

* Привязка компонента **Label** к полям элементам пользовательского интерфейса для быстрой навигации.
* Элементы пользовательского интерфейса **ComboBox** и **Slider**.
* Работа с контейнером **DockPanel**
* Работа со свойствами компонента **RichTextBox.**
* Работа с элементом **BackgroundWorker,** запуск задания в отдельном потоке.
* Встраивание компонентов WindowsForms посредством элемента **WindowsFormsHost**.
* Создание пользовательского элемента управления.
* Работа с графикой.
* Работа с диаграммами, посредством подключения сторонней библиотеки.

После выполнения лабораторной работы становится очевидно, что система WPF является мощным инструментом для создания приложений, позволяющая не только использовать базовый набор элементов, предоставляемых ToolBox, но и создавать собственные элементы расширяя функционал базовых.

Благодаря многопоточности C#, WPF приложения могут содержать несколько потоков для выполнения отдельных задач в фоне, не блокируя пользовательский интерфейс.