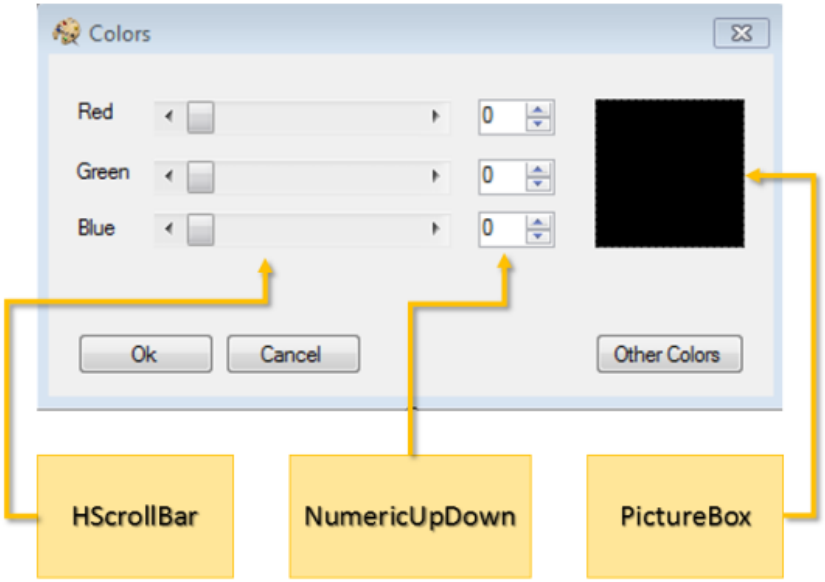
Лабораторная работа 13

Дополнение графического редактора

Цель работы: дополнить графический редактор отдельной формой выбора цвета.

Практическая часть

Завершающим штрихом для нашего графического редактора будет отдельная форма для выбора цвета пера. Выглядеть она будет следующим образом:



Для того, чтобы ее добавить, нужно зайти в *проект->добавить форму Windows.*

1. Необходимо разместить все компоненты, как показано на рисунке.

2. Для компонентов *HScrollBar* нужно установить свойство *Minimum* в 0, а *Maximum* в 255. Это будет диапазон изменения оттенков цветов. Так же выставляем *LargeChage* в единицу, это будет величина прокрутки при щелчке на полосе. Точно такие же значения свойств прописываем и для *NumericUpDown*, только вместо *LargeChage* будет свойство *Increment*.

3. Теперь нам нужно связать *HScrollBar* и *NumericUpDown*. Для этого в конструкторе новой формы пропишем следующий код, который свяжет эти 2 компонента через свойство *Tag*:

```
public ColorsForm(Color color)
{
	InitializeComponent();
	Scroll_Red.Tag = numeric_Red;
	Scroll_Green.Tag = numeric_Green;
	Scroll_Blue.Tag = numeric_Blue;
	numeric_Red.Tag = Scroll_Red;
	numeric_Green.Tag = Scroll_Green;
	numeric_Blue.Tag = Scroll_Blue;
	numeric_Red.Value = color.R;
	numeric_Green.Value = color.G;
	numeric_Blue.Value = color.B;
}
```

Для отличия между собой все 6 компонентов были переименованы.

4. Т.к. задача заключается в связывании двух компонентов *HScrollBar* и *NumericUpDown*, то воспользуемся событием *ValueChanged* у обоих компонентов.

   Для *HScrollBar*:

```
private void Scroll_Red_ValueChanged(object sender, EventArgs e)
{
	ScrollBar scrollBar = (ScrollBar)sender;
	NumericUpDown numericUpDown = (NumericUpDown)scrollBar.Tag;
	numericUpDown.Value = scrollBar.Value;
}
```

   Для NumericUpDown:

```
private void numeric_Red_ValueChanged(object sender, EventArgs e)
{
	NumericUpDown numericUpDown = (NumericUpDown)sender;
	ScrollBar scrollBar = (ScrollBar)numericUpDown.Tag;
	scrollBar.Value = (int)numericUpDown.Value;
}
```

Аналогично прописывается еще 2 события для оставшихся двух *HScrollBar* и 2 события для двух оставшихся *NumericUpDown*.

5. Теперь нам потребуется одна глобальная для класса переменная *colorResult*:

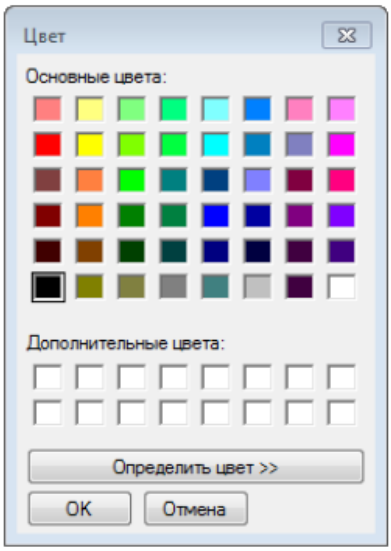
```
Color colorResult;
```

А также отдельная функция (назовем ее *UpdateColor*), которая будет смешивать цвет на основании положения ползунков и закрашивать данным цветом *PictureBox*. Т.к. в данном *PictureBox* мы не будем рисовать, а будем использовать его лишь для определения цвета пера, то инициализировать его свойство Image не придется, как это делали в предыдущей части лабораторной работы. Для *UpdateColor* не принципиально откуда брать значения – из *HScrollBar* или из *NumericUpDown*, т.к. они теперь связаны между собой. Для примера взят именно *HScrollBar*:

```
private void UpdateColor()
{
      colorResult = Color.FromArgb(Scroll_Red.Value,
      Scroll_Green.Value,
      Scroll_Blue.Value);
     picResultColor.BackColor = colorResult;
}
```

Вызывать ее можно либо в *HScrollBar*, либо в NumericUpDown, т.к. опять же не имеет значения, ведь после 4 пункта мы связали эти 2 компонента. Выбирается любой компонент и для всех трех его цветов вызывается функция *UpdateColor()* в любом месте.

Примечание 1. В .Net есть готовое решение по выбору цвета. Выглядит оно следующим образом:



Данное решение называется ColorDialog. Будем вызывать его по кнопке Other Colors. Помимо того, что будет появляться это окно, нужно будет еще привязать RGB выбранного цвета к в HScrollBar и NumericUpDown. Для этого будет служить следующий код:

```
private void buttonOther_Click(object sender, EventArgs e)
{
      ColorDialog colorDialog = new ColorDialog();
      if (colorDialog.ShowDialog() == DialogResult.OK)
      {
      Scroll_Red.Value = colorDialog.Color.R;
      Scroll_Green.Value = colorDialog.Color.G;
      Scroll_Blue.Value = colorDialog.Color.B;
      colorResult = colorDialog.Color;
      UpdateColor();
     }
}
```

6. Теперь переходим к связке двух форм. Связывать их будет лишь один параметр, который будет передаваться из формы 2 по нажатию на кнопку ***Ok*** в форму 1 – это созданный цвет пера *colorResult*. Существует много вариантов по передаче данных из одной формы в другую внутри одного проекта, разберем некоторые из них:

1. **Изменение модификатора доступа**

В Form2 установить модификатор доступа для контрола/поля public
В любом месте Form1

Код C#

```
1  Form2 f = new Form2();
2  f.ShowDialog();
3  this.textBox1.Text = f.textBox1.Text;
```

+Самый быстрый в реализации и удобный способ
- Противоречит всем основам ООП
- Возможна передача только из более поздней формы в более раннюю
- Форма f показывается только с использованием ShowDialog(), т.е. в первую форму управление вернется только по закрытию второй. Избежать этого можно, сохранив ссылку на вторую форму в поле первой формы.

2. **Использование открытого свойства/метода. Способ очень похож на первый**

В классе Form2 определяем свойство (или метод)

Код C#

```
1  public string Data
2  {
3     get
4     {
5        return textBox1.Text;
6     }
7  }
```

В любом месте Form1

Код C#

```
1  Form2 f = new Form2();
2  f.ShowDialog();
3  this.textBox1.Text = f.Data;
```

+ Противоречит не всем основам ООП
- Минусы те же

3. **Передача данных в конструктор Form2**

Изменяем конструктор Form2

Код C#

```
1  public Form2(string data)
2  {
3     InitializeComponent();
4     //Обрабатываем данные
5     //Или записываем их в поле
6     this.data = data;
7  }
8  string data;
```

А создаем форму в любом месте Form1 так:

Код C#

```
1  Form2 f = new Form2(this.textBox1.Text);
2  f.ShowDialog();
3  //Или f.Show();
```

+ Простой в реализации способ
+ Не нарушает ООП
- Возможна передача только из более ранней формы в более позднюю

4. Передача ссылки в конструктор

Изменяем конструктор Form2

Код C#

```
1  public Form2(Form1 f1)
2  {
3     InitializeComponent();
4     //Обрабатываем данные
5     //Или записываем их в поле
6     string s = f1.textBox1.Text;
7  }
```

А создаем форму в любом месте Form1 так, т.е. передаем ей ссылку на первую форму

Код C#

```
1  Form2 f = new Form2(this);
2  f.ShowDialog();
3  //Или f.Show();
```

+ Доступ ко всем открытым полям/функциям первой формы
+ Передача данных возможна в обе стороны
- Нарушает ООП

5. Используем свойство 'родитель'

При создании второй формы устанавливаем владельца

Код C#

```
1  Form2 f = new Form2();
2  f.Owner = this;
3  f.ShowDialog();
```

Во второй форме определяем владельца

Код C#

```
1  Form1 main = this.Owner as Form1;
2  if(main != null)
3  {
4     string s = main.textBox1.Text;
5     main.textBox1.Text = "OK";
6  }
```

+ Доступ ко всем открытым полям/функциям первой формы
+ Передача данных возможна в обе стороны
+ Не нарушает ООП

6. Используем отдельный класс

Создаем отдельный класс, лучше статический, в основном namespace, т.е. например в файле Program.cs

Код C#

```
1  static class Data
2  {
3      public static string Value { get; set; }
4  }
```

Его открытые свойства/методы доступны из любой формы.

Код C#

```
1  Data.Value = "111";
```

+ Самый удобный способ, когда данные активно используются несколькими формами.