## ЛАБОРАТОРНАЯ РАБОТА 1. Введение в WPF (часть 1). Контейнерные элементы управления

Windows Presentation Foundation (WPF) - это система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем. С помощью WPF можно создавать широкий спектр как автономных, так и размещенных в браузере приложений.

В основе WPF лежит векторная система отрисовки, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования. WPF расширяет базовую систему полным набором функций разработки приложений, в том числе Язык XAML (Extensible Application Markup Language), элементами управления, привязкой данных, макетом, двухмерный- и трехмерный-графикой, анимацией, стилями, шаблонами, документами, мультимедиа, текстом и оформлением. WPF входит в состав Microsoft .NET Framework и позволяет создавать приложения, включающие другие элементы библиотеки классов .NET Framework.

**Программирование с использованием WPF**

WPF существует в качестве подмножества типов .NET Framework, которые занимают большую часть в пространстве имен System.Windows. Пользователи, которые ранее создавали приложения с помощью .NET Framework, используя такие управляемые технологии, как ASP.NET и Windows Forms, должны быть знакомы с основами программирования WPF; создание экземпляров классов, задание свойств, вызов методов и обработка событий осуществляется аналогично.

Для поддержки некоторых более мощных возможностей WPF и для упрощения процесса программирования WPF включает дополнительные программные конструкции, которые расширяют свойства и события: *свойства зависимостей* и *перенаправленные события*.

**Разметка и код программной части**

В WPF дополнительно совершенствуется процесс программирования для разработки клиентских приложений Windows. Одним очевидным усовершенствованием является возможность разрабатывать приложения с помощью *разметки* и *кода программной части*, с которыми разработчики ASP.NET должны быть уже знакомы. Разметка Язык XAML обычно используется для реализации внешнего вида приложения при реализации его поведения с помощью управляемых языков программирования (кода программной части). Это разделение внешнего вида и поведения имеет следующие преимущества:

* Затраты на разработку и обслуживание снижаются, так как разметка определенного внешнего вида тесно не связана с кодом определенного поведения.
* Разработка более эффективна, так как разработчики, реализующие внешний вид приложения, могут это делать одновременно с разработчиками, реализующими поведение приложения.
* Для реализации и совместного использования разметки Язык XAML применяется множество средств конструирования, чтобы удовлетворить требованиям участников разработки приложений.
* Глобализация и локализация для приложений WPF существенно упрощены.

**Разметка**

Язык XAML - это основанный на XML язык разметки, который используется для декларативной реализации внешнего вида приложения. Обычно он используется для создания окон, диалоговых окон, страниц и пользовательских элементов управления, а также для их заполнения элементами управления, фигурами и графикой.

В следующем примере с помощью Язык XAML реализуется внешний вид окна, содержащего одну кнопку.

<Window

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

Title="Window with Button"

Width="250" Height="100">

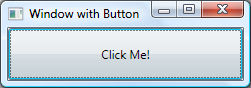
<!-- Add button to window -->

<Button Name="button">Click Me!</Button>

</Window>

В частности, этот Язык XAML-код определяет окно и кнопку с помощью элементов Window и Button, соответственно. Каждый элемент настраивается с помощью атрибутов, таких как атрибут Title элемента Window, в котором задается текст заголовка окна. Во время выполнения WPF преобразует элементы и атрибуты, определенные в разметке, в экземпляры классов WPF. Например, элемент Window преобразуется в экземпляр класса [Window](http://msdn.microsoft.com/ru-ru/library/system.windows.window(v=vs.110).aspx), свойство [Title](http://msdn.microsoft.com/ru-ru/library/system.windows.window.title(v=vs.110).aspx) которого является значением атрибута Title.

На следующем рисунке показан пользовательский интерфейс, который задается с помощью Язык XAML в предыдущем примере.



Поскольку Язык XAML основан на XML, Пользовательский интерфейс, который формируется с его помощью, организуется в виде иерархии вложенных элементов, называемой *деревом элементов*. Дерево элементов предоставляет логичный и интуитивно понятный способ для создания и управления пользовательских интерфейсов.

**Код программной части**

Приложение в основном предназначено для реализации функциональных возможностей, которые отвечают на взаимодействия с пользователем, включая обработку событий (например, нажатие меню, панели инструментов или кнопки) и вызов бизнес-логики и логики доступа к данным в ответ на события. В WPF такое поведение обычно реализуется в коде, который связан с разметкой. Этот тип кода называется кодом программной части. В следующем примере показан код программной части и обновленная разметка из предыдущего примера.

namespace SDKSample

{

public partial class AWindow : Window

{

public AWindow()

{

// InitializeComponent call is required to merge the UI

// that is defined in markup with this class, including

// setting properties and registering event handlers

InitializeComponent();

}

void button\_Click(object sender, RoutedEventArgs e)

{

// Show message box when button is clicked

MessageBox.Show("Hello, Windows Presentation Foundation!");

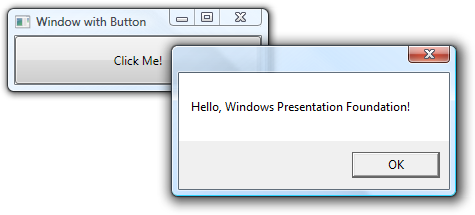
}

}

}

В этом примере код программной части реализует класс, производный от класса [Window](http://msdn.microsoft.com/ru-ru/library/system.windows.window(v=vs.110).aspx). Атрибут x:Class связывает разметку с классом кода программной части. Метод InitializeComponent вызывается из конструктора класса кода программной части, чтобы объединить пользовательский интерфейс, определенный в разметке, с этим классом. (InitializeComponent генерируется при построении приложения, поэтому нет необходимости реализовывать его вручную.) Комбинация x:Class и InitializeComponent позволяет гарантировать правильную инициализацию созданной реализации. Класс кода программной части также реализует обработчик событий для события кнопки [Click](http://msdn.microsoft.com/ru-ru/library/system.windows.controls.primitives.buttonbase.click(v=vs.110).aspx). При нажатии кнопки обработчик событий показывает окно сообщения с помощью метода [MessageBox.Show](http://msdn.microsoft.com/ru-ru/library/system.windows.messagebox.show(v=vs.110).aspx).

На следующем рисунке показан результат нажатия этой кнопки.



**ЗАДАНИЕ**

**# 1A.**

Создайте интерфейс WPF-приложения, используя Grid, Canvas или StackPanel. Разместите элементы управления внутри макета.

Создайте приложение, моделирующее работу кофейного автомата с помощью автоматного программирования. В автомат опускается произвольное количество монет любого достоинства. Автомат выдает сдачу в зависимости от сделанного заказа. Автомат содержит группы кнопок:

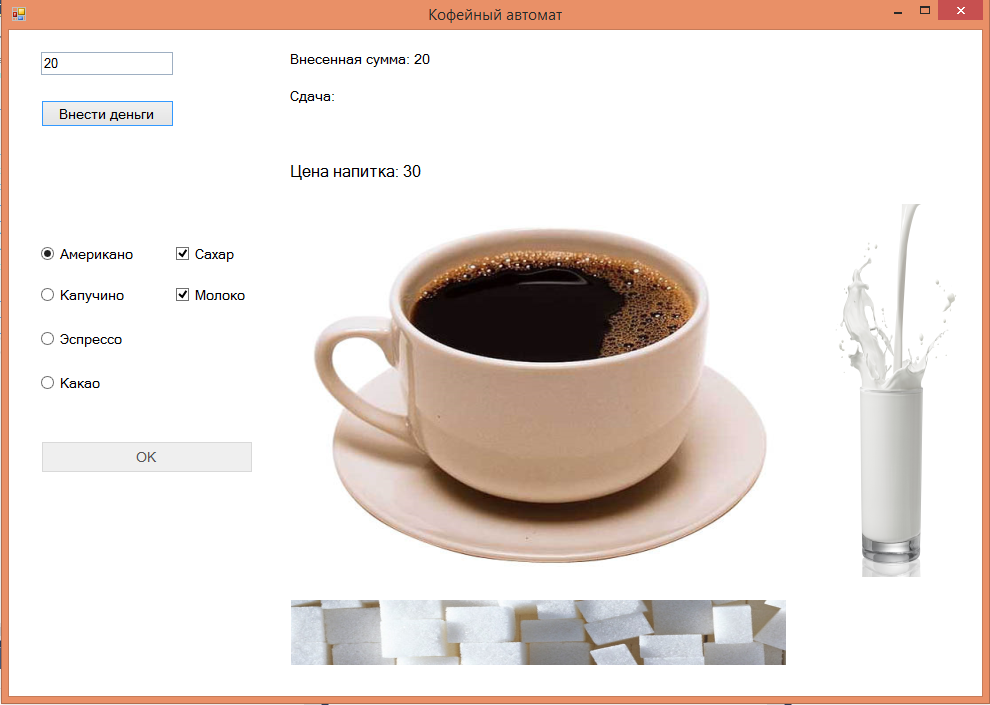
1. «американо», «капучино», «эспрессо», «какао»;
2. «с сахаром», «без сахара»;
3. «с молоком», «без молока».

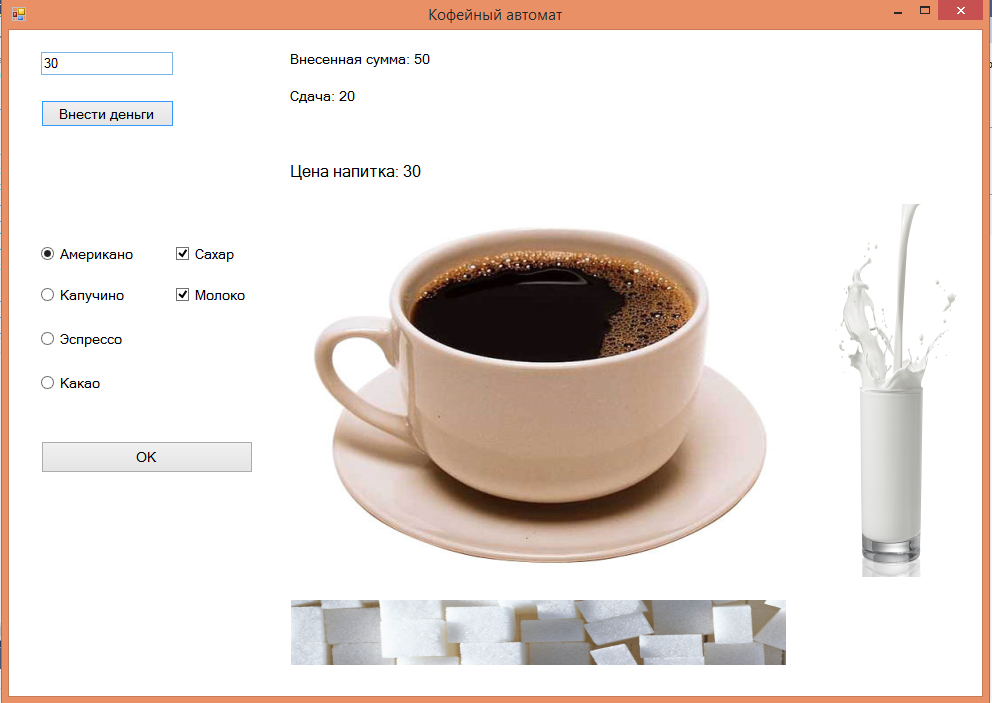
При выборе кнопки напитка, в Image отображается выбранный объект.

Имеется кнопка «ОК», которая доступна после выбора конфигурации и получения автоматом денег.

После нажатия кнопки «ОК» выводится сообщение о готовности и сумма сдачи. Счет в Label обнуляется (первоначальная форма).

**Не забудьте использовать КЛАССЫ для напитков.**





**# 1B.**

Создайте WPF-приложение, моделирующее работу лифта. Лифт – это устройство, которое может иметь различные состояния. Например:

1. N-ый этаж: двери закрыты;
2. N-ый этаж: двери открыты.

Любая команда переводит лифт из одного состояния в другое (например, использовать **enum** c состояниями – ВВЕРХ, ВНИЗ, ОТКРЫТО, ЗАКРЫТО). Спускаясь вниз, лифт не подбирает пассажиров.

Нельзя:

* двигаться с открытыми дверями,
* подниматься выше, когда лифт находится на последнем этаже.
* опускаться ниже первого этажа.

Создать **класс** для Лифта, определяющий функции его передвижения. При запуске программы пользователь должен ввести количество этажей в здании.