Лекция 6:
Динамические массивы
Windows Forms
Потоки

Обобщенные типы

Обобщенные типы - это типы с параметрами. Часть системы типов .NET Framework, которая позволяет определять тип.

Простые классы Cache и Generic-класс Cache

```
public class CacheString {
    private string message = "";
    public void add(string message){
        this.message = message;
    }
    public string get(){
        return this.message;
    }
}
```

```
public class CacheShirt {
    private Shirt shirt;

    public void add(Shirt shirt){
        this.shirt = shirt;
    }

    public Shirt get(){
        return this.shirt;
    }
}
```

```
public class CacheAny <T>{

    private T t;

    public void add(T t){
        this.t = t;
    }

    public T get(){
        return this.t;
    }
}
```

Синтаксис Generic-типов

```
class Gen<T, U>
{
public T t;
public U u;
public Gen(T _t, U _u)
    {
        t = _t;
        u = _u;
    }
}
```

- **Класс Gen имеет два члена типа *T* и *U*.**
  - Код, использующий этот класс, определит типы для *T* и *U*. В зависимости от того, как класс Gen используется в коде, могут иметь тип string, int, пользовательский тип или другую их комбинацию.

Применение обобщений

- **Для использования обобщений нужно указать его тип.**

```
// Add two strings using the Gen class

Gen<string, string> ga = new Gen<string, string>("Hello, ",
"World!");
Console.WriteLine(ga.t + ga.u);

// Add a double and an int using the Gen class

Gen<double, int> gb = new Gen<double, int>(10.125, 2005);
Console.WriteLine(gb.t + gb.u);
```

Обобщенные типы (Generic Types)

- **Преимущества:**
  - производительность;
  - вместо использования объектов можно использовать класс List<T> из пространства имен System.Collection.Generic, который позволяет определить тип элемента при создании коллекции.

```
List<int> list = new List<int>();
list.Add(44);//нет упаковки – элементы
             //сохраняются в List<int>
int il=list[0];// распаковка не нужна
foreach (int i2 in list)
{    Console.WriteLine(i2);
     }
```

- **Преимущества:**
  - безопасность типов:

    например, когда в классе ArrayList сохраняются объекты, то в коллекцию могут быть вставлены объекты различных типов.

```
ArrayList list = new ArrayList();
list.Add(44); // вставка целого
list.Add("mystring"); // вставка строки
list.Add(new MyClass ()) ; // вставка объекта
```

```
foreach (int i in list) {
Console.WriteLine(i);
}// возникнет ошибка во время выполнения!!!
```

```
List<int> list = new List<int>();
list.Add(44) ;
list.Add("mystring");    // ошибка компиляции
list.Add(new MyClass());    // ошибка компиляции
```

Использование ограничений

- **Ограничения — позволяют определить требования к типам, которыми разрешено заменять обобщения в коде.**
- **Обобщения поддерживают четыре типа ограничений:**
  - По интерфейсу.
  - По базовому классу.
  - По конструктору.
  - По ссылочному или значимому типу.
- **Для применения ограничений к обобщению используется секция `where`.**

```
// класс обобщения может использоваться только типами,
// реализующими интерфейс IComparable
class CompGen<T>
where T : IComparable
{
public T t1;
public T t2;
public CompGen(T _t1, T _t2)
    {
        t1 = _t1;
        t2 = _t2;
    }
public T Max()
        {
    if (t2.CompareTo(t1) < 0)
        return t1;
    else
        return t2;
        }
}
```

Динамические массивы

Динамическим называется **массив**, размер которого может меняться во время исполнения программы.

Классы-коллекции

- Библиотека .NET Framework содержит большой набор классов-коллекций, которые используются при работе с наборами элементов.
- Классы-коллекции из пространства имен System.Collections поддерживают слабо типизированные коллекции, элементы которых имеют тип System.Object. Эти коллекции поддерживаются всеми версиями .NET Framework, начиная с 1.0.
- В версиях .NET Framework 2.x и выше в пространстве имен System.Collections.Generic определены интерфейсы и классы обобщенных коллекций, которые дают возможность создавать строго типизированные коллекции.

Классы-коллекции

- В версиях 1.x было определено небольшое число строго типизированных коллекций в пространстве имен System.Collection.Specialized ,
- а в пространстве имен System.Collections определены абстрактные классы, которые можно было использовать как базовые для создания пользовательских типизированных (strong typed) коллекций.
  - CollectionBase : IList, ICollection, IEnumerable
  - DictionaryBase : IDictionary, ICollection, IEnumerable
  - ReadOnlyCollectionBase : ICollection, IEnumerable
  - NameObjectCollectionBase : ICollection, IEnumerable, ISerializable, IDeserializationCallback
- В версиях 2.x и выше определены обобщенные классы-коллекций в пространстве имен System.Collections.Generic – это прямые аналоги обычных типов-коллекций.

Интерфейс ICollection

- Интерфейс ICollection реализуют все необобщенные и некоторые обобщенные классы-коллекции

```
public interface ICollection : IEnumerable {
  int Count {get;}        - число элементов в коллекции
  bool IsSynchronized {get;}  - информация о том, является ли доступ
          синхронизированным ( thread-safe)
  object SyncRoot {get;}  - дает доступ к объекту синхронизации
  void CopyTo( Array array, int index );
                - копирует элементы коллекции в массив                    Array
...
}
```

```
public interface ICloneable {
  object Clone();     - создает новый объект-копию
}
```

Интерфейс IList

```
public interface IList : ICollection, IEnumerable {
    object this[ int index ] {get; set;}
    bool IsFixedSize {get;}
    bool IsReadOnly {get;}
    int Add( object value );
    void Clear();
    bool Contains( object value );
    int IndexOf( object value );
    void Insert( int index, object value );
    void Remove( object value );
    void RemoveAt( int index );
}
```

- В коллекции, реализующей интерфейс IList:
  - определена операция индексирования с целым индексом;
  - в коллекцию можно вставлять элементы.
- IList реализован в коллекциях, к элементам которых можно обращаться по индексу, в коллекцию можно добавлять элементы (реализован в ArrayList).

Интерфейс ICollection<T> и IList<T>

- Обобщенный интерфейс ICollection<T>

```
public interface
ICollection<T> : IEnumerable<T>, IEnumerable
{    int Count { get; }
     bool IsReadOnly { get; }
     void Add (T item);
     void Clear ();
     bool Contains ( T item);
     void CopyTo (T[] array, int arrayIndex);
     boolean Remove ( T item);
}
```

- Обобщенный интерфейс IList<T>

```
public interface
IList<T> : ICollection<T>, IEnumerable<T>, IEnumerable
{    T this [int index] { get; set; }
     int IndexOf (T item);
     void Insert (int index, T item);
     void RemoveAt (int index);
}
```

Интерфейсы для итераторов

- Следующие обобщенные интерфейсы определены в пространстве имен System.Collections.Generic.

```
public interface IEnumerable<T> : IEnumerable
{    IEnumerator<T> GetEnumerator ();}

public interface IEnumerator<T> : IDisposable, IEnumerator
{    T Current { get; } }
```

- Следующие обычные(необобщенные) интерфейсы определены в пространстве имен System.Collections.

```
public interface IEnumerator {
    object Current {get;}
    bool MoveNext();
    void Reset();
}

public interface IEnumerable {
    IEnumerator GetEnumerator();
}
```

Классы-коллекции из пространства имен System.Collections

Класс	Интерфейсы	Описание
ArrayList	IList, ICollection, IEnumerable, ICloneable	массив элементов с динамически изменяющимся размером
Queue	ICollection, IEnumerable, ICloneable	очередь FIFO (first in - first out)
Stack	ICollection, IEnumerable, ICloneable	очередь LIFO (last in - first out)
BitArray	ICollection, IEnumerable, ICloneable	массив бит (любой длины)

Классы-коллекции из пространства имен System.Collections.Specialized

Класс	Интерфейсы	Описание
StringCollection	IList, ICollection, IEnumerable	Типизированная коллекция строк (string), реализующая IList.
struct BitVector32	-	Эффективен для набора булевских переменных или небольших целых чисел

Обобщенные классы Queue<T> и Stack<T>

Класс	Интерфейсы	Описание
Queue<T>	IEnumerable<T>, ICollection, IEnumerable	Очередь. Обобщенная версия Queue.
Stack<T>	IEnumerable<T>, ICollection, IEnumerable	Стек. Обобщенная версия Stack.

Обобщенные классы List<T> и LinkedList<T>

Класс	Интерфейсы	Описание
List<T>	IList<T>, ICollection<T>, IEnumerable<T>, IList, ICollection, IEnumerable	Список объектов, к которым возможен доступ по целочисленному индексу. Обобщенная версия ArrayList. Элементы хранятся в массиве. Быстро добавляются элементы в конец, поиск и вставка более медленные.
LinkedList<T>	ICollection<T>, IEnumerable<T>, ICollection, IEnumerable, ISerializable, IDeserializationCallback	Связанный список. Нет аналога в обычных коллекциях. Реализован в виде цепочки динамически размещаемых объектов. Быстрая операция вставки. Замедляет работу сборщика мусора . При больших списках затраты, связанные с каждым узлом.

Обобщенные классы List<T> и LinkedList<T>

Класс	Интерфейсы	Описание
List<T>	IList<T>, ICollection<T>, IEnumerable<T>, IList, ICollection, IEnumerable	Список объектов, к которым возможен доступ по целочисленному индексу. Обобщенная версия ArrayList. Элементы хранятся в массиве. Быстро добавляются элементы в конец, поиск и вставка более медленные.
LinkedList<T>	ICollection<T>, IEnumerable<T>, ICollection, IEnumerable, ISerializable, IDeserializationCallback	Связанный список. Нет аналога в обычных коллекциях. Реализован в виде цепочки динамически размещаемых объектов. Быстрая операция вставки. Замедляет работу сборщика мусора . При больших списках затраты, связанные с каждым узлом.

Классы-словари (хэш-таблицы)

- Классы-словари (хэш-таблицы) используются для работы с множеством пар <ключ, значение>.
- В паре <ключ - значение> ключ обеспечивает косвенную ссылку на данные (значение) :
  - ключ не может иметь значение null, value может иметь значение null;
  - коллекция не может содержать две пары с одинаковым ключом.
- С помощью хэш-функции ключ преобразуется в целочисленное значение (хэш-код) из некоторого диапазона 0, ... N-1. Хэш-код используется как индекс в таблице (hash table), в которой хранятся прямые или косвенные ссылки на данные (значения).
- Классы-словари реализуют интерфейсы IDictionary или IDictionary <TKey,TValue>.

Структуры для элементов коллекций

- В необобщенных коллекциях-словарях каждый элемент (пара ключ-значение) хранится в объекте DictionaryEntry.

```
public struct DictionaryEntry {
  public DictionaryEntry( object key, object value );
  public object Key {get; set;}
  public object Value {get; set;}
}
```

- В обобщенных коллекциях-словарях каждый элемент (пара ключ-значение) хранится в объекте KeyValuePair < TKey,TValue>.

```
public struct  KeyValuePair < TKey,TValue>
{  public KeyValuePair ( TKey key, TValue value );
   public TKey Key { get; }
   public TValue Value { get; }
}
```

Интерфейс IDictionary

- Интерфейс IDictionary реализуют классы:
  - Hashtable
  - HybridDictionary
  - ListDictionary.
  - DictionaryBase
  - SortedList

```
public interface IDictionary : ICollection, IEnumerable {
  bool IsFixedSize {get;}
  bool IsReadOnly {get;}
  object this[ object key] {get; set;}
  ICollection Keys {get;}
  ICollection Values {get;}
  void Add ( object key, object value);
  void Clear();
  void Remove( object key );
  bool Contains ( object key );
  IDictionaryEnumerator GetEnumerator();
}
```

Интерфейс IDictionary <TKey, TValue>

```
public interface IDictionary < TKey,TValue> :
         ICollection < KeyValuePair < TKey,TValue>>,
         IEnumerable < KeyValuePair < TKey,TValue>>,
         IEnumerable
{ TValue this [ TKey key];
  ICollection <TKey> Keys { get; }
  ICollection <TValue> Values { get; }
  void Add ( TKey key, TValue value);
  bool ContainsKey ( TKey key);
  bool Remove ( TKey key);
  bool TryGetValue (TKey key, out TValue value);
}
```

Хэш-функция

- Хэш-функция используется для быстрой генерации числа (hash code), отвечающего значению объекта. По умолчанию используется функция GetHashCode(). Для корректной работы классов-словарей для хэш-функции GetHashCode() должно быть выполнено:
- для двух совпадающих объектов одного и того же типа хэш-функция должна возвращать одно и тоже значение;
- реализация GetHashCode() не должна бросать исключения;
- если в производном классе переопределен метод GetHashCode(), в нем также должен быть переопределен метод Equals() так, чтобы два равных объекта имели одно и то же значение хэш-кода.
- Метод класса Object может использоваться как хэш-функция, если равенство объектов понимается как равенство ссылок.

```
public virtual int GetHashCode();
```

- Реализация GetHashCode() из класса String возвращает уникальный хэш-код для каждого значения string.

Классы-словари из пространства имен System.Collections

Класс	Интерфейсы	Описание
HashTable	IDictionary, ICollection, IEnumerable, ISerializable, IDeserializationCallback, ICloneable	словарь (пары ключ-значение) с быстрым доступом по ключу (упорядочен по хэш-коду ключа)
SortedList	IDictionary, ICollection, IEnumerable, ICloneable	таблица с быстрым доступом по ключу или индексу (гибрид Array и Hashtable)

Классы-словари из пространства имен System.Collections.Specialized

Класс	Интерфейсы	Описание
StringDictionary	IEnumerable	Коллекция пар < string , string >.
ListDictionary	IDictionary, ICollection, IEnumerable	Коллекция, реализующая IDictionary как односвязный список. Эффективнее Hashtable, если число элементов не превышает 10.
HybridDictionary	IDictionary, ICollection, IEnumerable	Коллекция, которая реализуется как ListDictionary для небольшого числа элементов и преобразуется в Hashtable при увеличении числа элементов.

Обобщенные классы-словари

Класс	Описание
Dictionary <TKey,TValue>	Коллекция пар ключ-значение. Обобщенная версия Hashtable. Реализация в виде хэш-таблицы. Быстрый поиск, удаление и добавление элементов для больших коллекций.
SortedDictionary <TKey,TValue>	Коллекция пар ключ-значение, отсортированных по ключу на основе реализации IComparer. Реализация в виде сбалансированного дерева. Быстрый поиск, более медленная вставка. Нет аналога в обычных коллекциях.
SortedList <TKey,TValue>	Коллекция пар ключ-значение, отсортированных по ключу. Обобщенная версия SortedList. Данные хранятся в виде двух отдельных массивов для ключей и значений, для обоих поддерживается порядок.

Коллекция HashSet<T>

- Начиная с версии 3.5 .NET Framework поддерживается неупорядоченная коллекция из несовпадающих элементов

```
public class HashSet<T> : ICollection<T>, IEnumerable<T>,
          IEnumerable, ISerializable, IDeserializationCallback
```

- Методы класса позволяют выполнять быстрые операции объединения, пересечения, проверки совпадения двух коллекций.

```
public bool SetEquals( IEnumerable<T> other );         // true, если
                                  // коллекции совпадают;
publi bool Overlaps( IEnumerable<T> other );          // true, если
              // в коллекциях есть хотя бы один совпадающий элемент;
public void ExceptWith( IEnumerable<T> other );       // удаляет из
     // коллекции все элементы, которые есть в другой коллекции;
public void UnionWith( IEnumerable<T> other );        // добавляет в
     // коллекцию элементы из другой коллекции (только несовпадяающие);
public void IntersectWith( IEnumerable<T> other );    // оставляет в
     // коллекции только элементы, которые есть и в другой коллекции;
```

Работа с итератором

```
using System;
using System.Collections.Generic;
namespace ConsoleApplication1{
  class Program{
    static void Main(){
       // Создадим два множества
       SortedSet<char> ss = new SortedSet<char>();
       SortedSet<char> ss1 = new SortedSet<char>();
       ss.Add('A'); ss.Add('B'); ss.Add('C');
              ss.Add('Z');
       ShowColl(ss, "Первая коллекция: ");
       ss1.Add('X'); ss1.Add('Y'); ss1.Add('Z');
       ShowColl(ss1, "Вторая коллекция");
       ss.UnionWith(ss1);
       ShowColl(ss,"Объединение множеств:");
       ss.ExceptWith(ss1);
       ShowColl(ss, "Вычитание множеств");
       //Сравнение множеств???
       if (CmpSortedSet)
                  Console.WriteLine("Мн-во ss = мн-ву
       else
                Console.WriteLine(" Мн-во ss != мн-ву ss1");
}
```

```
bool CmpSortedSet(SortedSet<char> a,
                                    Sorte
dSet<char> b){
    if (a == b) return true;
    if ((a == null) || (b == null) ||
              (a.Count != b.Count))
                          return false;
    SortedSet<char>.Enumerator
                                 e1 =
a.GetEnumerator(),
                     e2 =
b.GetEnumerator();
    while (e1.MoveNext() &&
                                       e
2.MoveNext())
        if (e1.Current != e2.Current)
            return false;
    return true;
}
```

```
static void ShowColl
(SortedSet<char> ss, string s){
        Console.WriteLine(s);
        foreach (char ch in ss)
           Console.Write(ch + " ");
         Console.WriteLine("\n");
}
```

Введение в разработку форм для Windows

Выбор между формами Windows Forms и веб-формами

Выбор нужной технологию на основе назначения приложения

Пример.

- при создании веб-узла электронной торговли с общим доступом к нему в Интернете целесообразно разрабатывать приложение с помощью страниц с веб-формами.
- при построении интенсивно работающего быстродействующего приложения, для которого необходимо использовать все функциональные возможности клиентского компьютера (например, офисного приложения) лучше пользоваться формами Windows Forms.

Инструменты Visual Studio для разработки приложений Windows

- Визуальные конструкторы для Windows Forms с элементами управления для перетаскивания.
- Визуальные конструкторы для Windows Presentation Foundation.
- Оптимизированные редакторы кода, включающие в себя завершение операторов, проверку синтаксиса и другие возможности IntelliSense.
- Встроенные средства компиляции и отладки.
- Средства управления проектом, позволяющие создавать и управлять файлами приложения, в том числе локальным развертыванием, через интрасеть или Интернет.

Архитектура Windows Forms

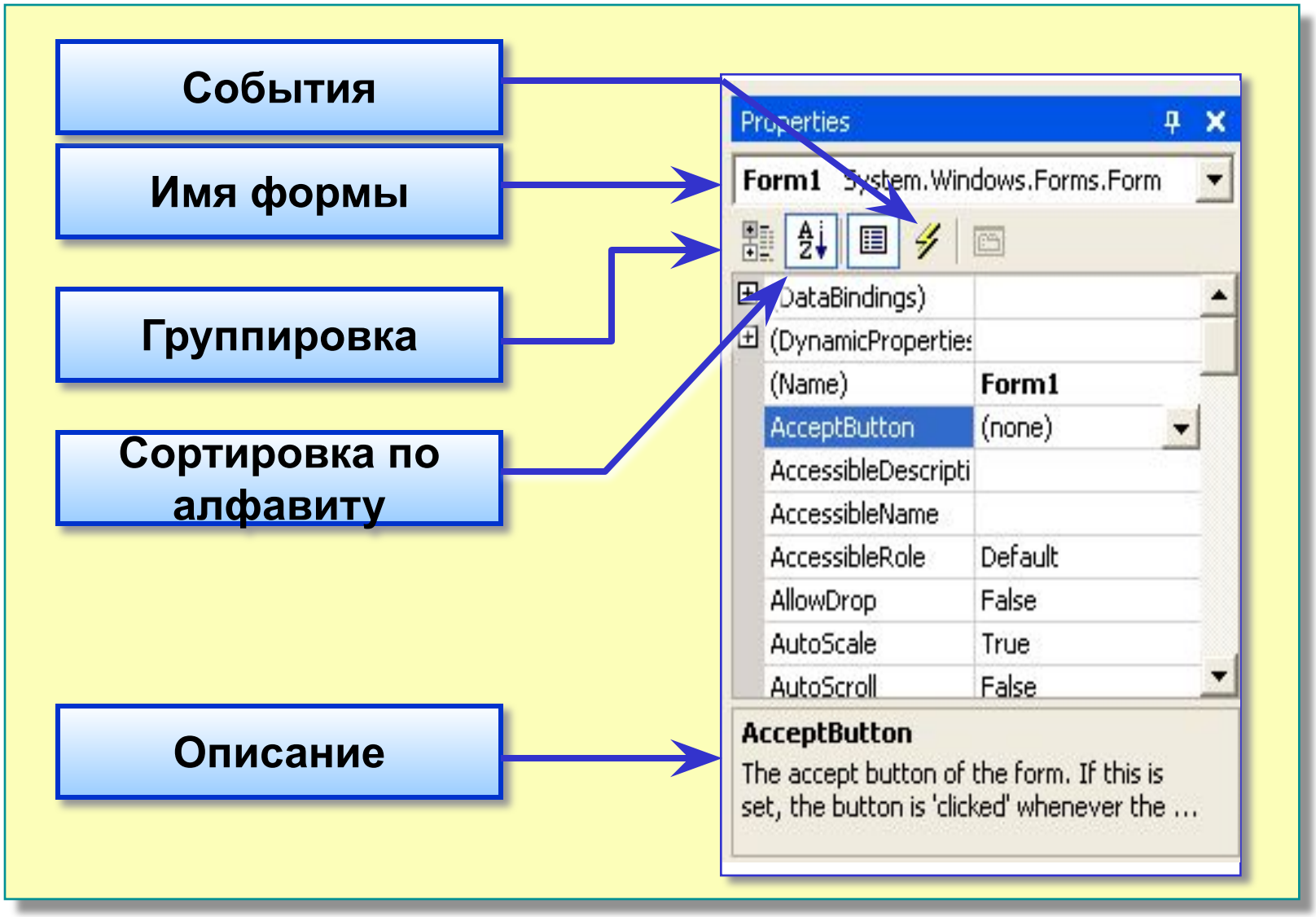
- **System.Drawing**
  - доступ к базовой функциональности графики GDI+,
  - более совершенная функциональность предоставляется в System.Drawing.Drawing2D, System.Drawing.Imaging, System.Drawing.Text.

http://msdn.microsoft.com/ru-ru/system.drawing.aspx

- **System.Windows.Forms**
  - содержит классы для создания Windows приложений,
  - доступ к стандартным элементам управления и компонентам

http://msdn.microsoft.com/ru-ru/system.windows.forms.aspx

Задание свойств форм



Изменение внешнего вида и поведения формы

- **Установка заголовка формы (свойство *Text)***

```
Forml.Text = "This is Form 1";
```

- **Установка типа границы (свойство *FormBorderStyle*)**

```
aForm.FormBorderStyle = FormBorderStyle.Fixed3D;
```

- **Настройка состояния формы при запуске (*WindowState):***
  - Normal, Minimized и Maximized
- **Изменение размера формы (свойство *Size)***

```
aForm.Size = new Size(300, 200);
```

- **Прозрачность и непрозрачность форм (свойство *Opacity*)**

```
aForm.Opacity = 0.5;
```

Настройка стартовой формы

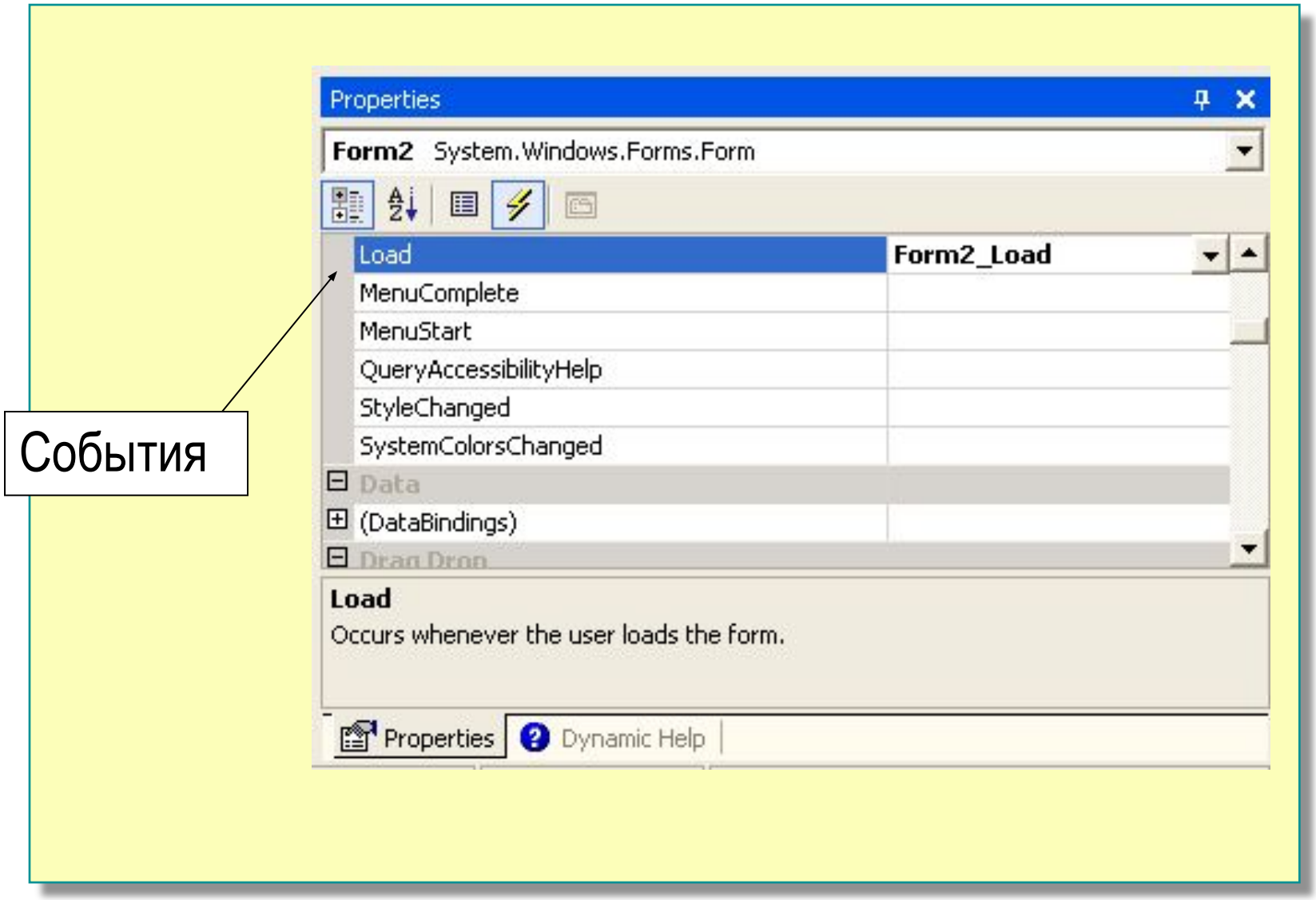
- Стартовая форма — это форма, которая при выполнении приложения загружается первой.
- Стартовый объект указывается в методе *Main.*
- По умолчанию этот метод расположен в классе *Program* и автоматически создается Visual Studio.
- Стартовый объект обозначен строкой:

```
Application.Run(new Form1());
```

- Как сделать стартовую форму невидимой (свойство *Visible*):

```
aForm.Visible = false;
```

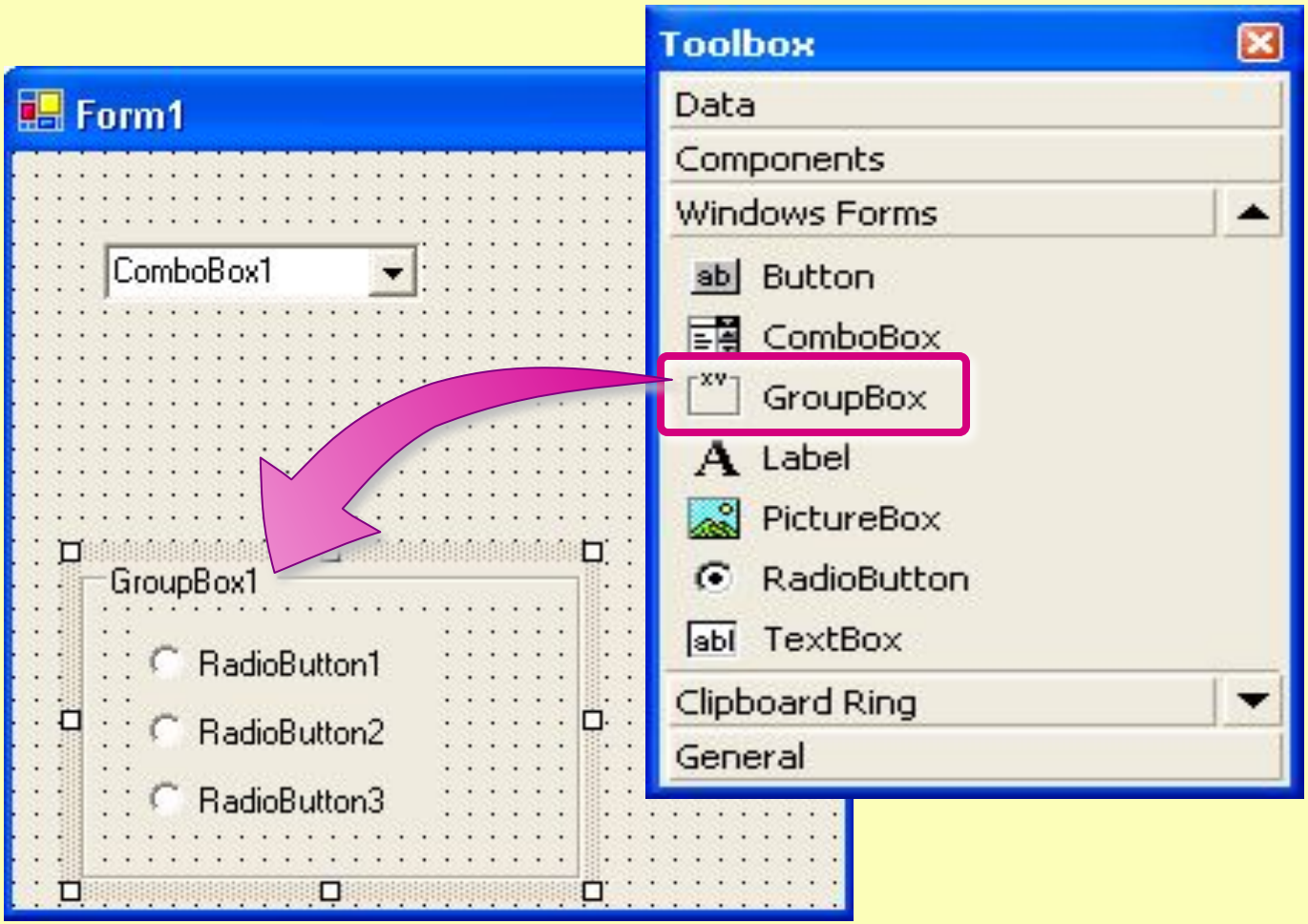
Обработка событий формы



Реализация обработчика события загрузки формы

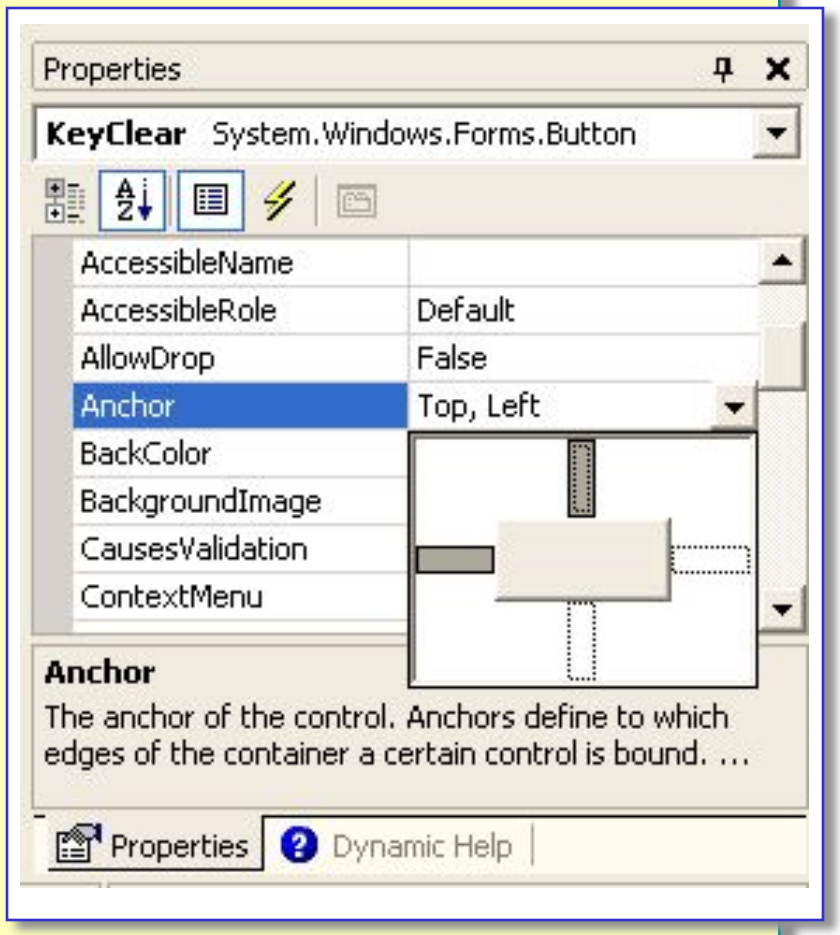
```
private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Стартовая форма";
  this.FormBorderStyle = FormBorderStyle.Fixed3D;
            this.Size = new Size(300, 200);
            this.Opacity = 0.8;
            this.TopMost = true;
 MessageBox.Show("Посмотрите на свойства формы",
"Внимание!!!");
        }
```

Добавление элементов управления на форму

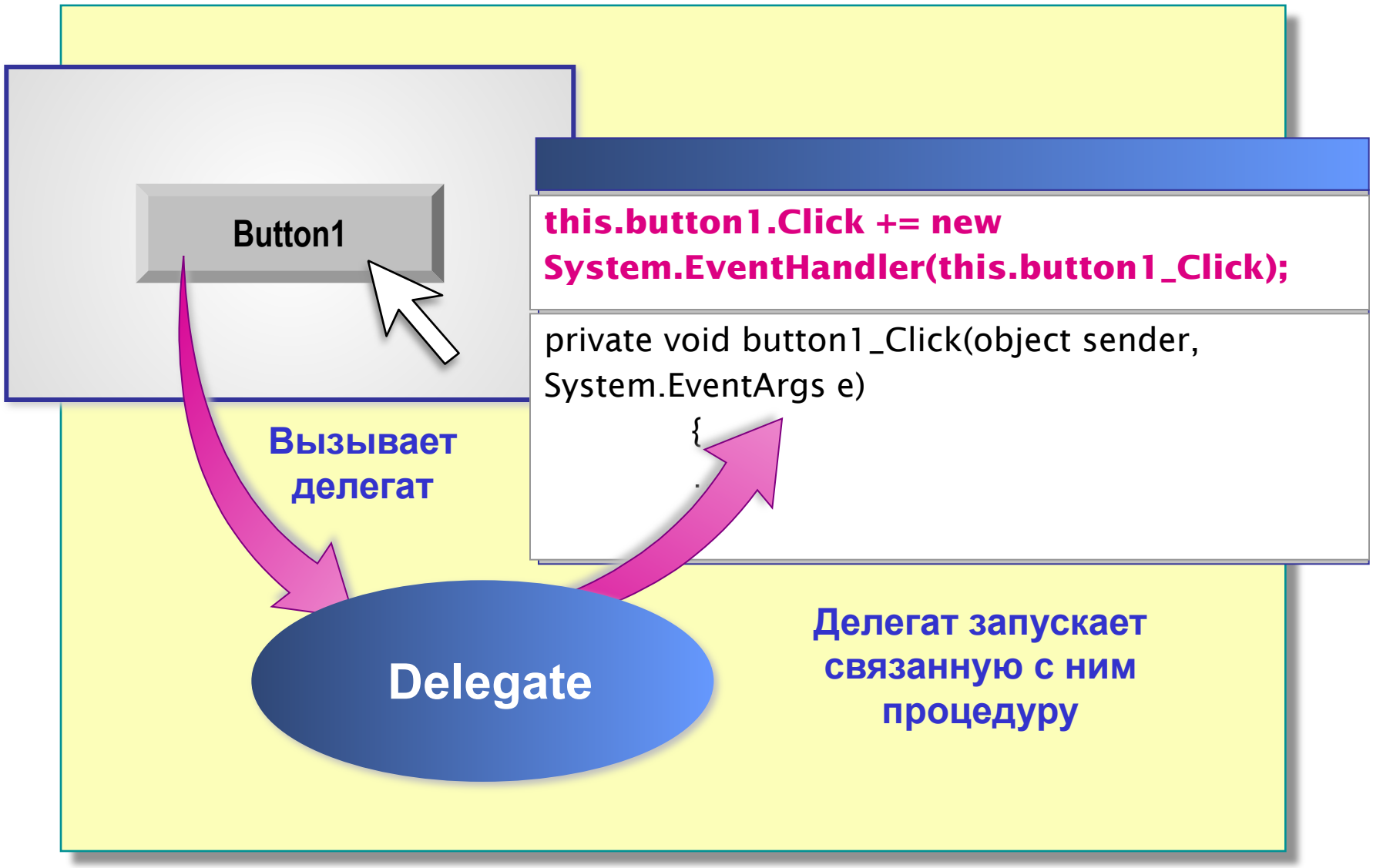


Привязка элемента управления к форме

- **Привязка**
  - Определяет, какие края элемента управления не меняют позицию по отношению к краям родительского контейнера
- **Для привязки элемента управления к форме**
  - Установите свойство **Anchor**
  - Значение по умолчанию: **Top**, **Left**
  - Другие возможные значения: **Bottom**, **Right**



Событийная модель в .NET Framework



Что такое делегаты?

- **Делегат**
  - Привязывает события к методам
  - Можно привязать к одному или нескольким методам
- **Когда событие регистрируется приложением**
  - ЭУ генерирует событие через вызов делегата для события
  - Делегат в свою очередь запускает связанный с ним метод

```
public delegate void EventHandler
                    (object sender, AlarmEventArgs e);
```

Как создавать обработчики событий

```
private void button1_Click(object sender,
                                        System.EventArgs e)
{
        MessageBox.Show("MyHandler received the event");

}
```

Как динамически добавлять и удалять обработчики событий

- **Для динамического связывания события (имя события - Click) с обработчиком события, используйте оператор +=**

```
this.button2.Click += new
        System.EventHandler(this.button1_Click);
```

- **Для динамического удаления связи события (Click) с обработчиком события используйте оператор -=**

```
this.button2.Click -= new
System.EventHandler(this.button1_Click);
```

Элементы управления, отображающие список

ListBox, **ComboBox, CheckedListBox**.

- отличаются внешним видом и функциональными возможностями,
- одинаково формируют и представляют списки данных и включают в себя коллекцию **Items**

1. Добавьте на форму ЭУ **ListBox**
2. Добавьте элементы **ListBox**, используя коллекцию **Items**
3. Задайте свойства ЭУ **ListBox**

Элементы управления, отображающие список

- **Для добавления элемента используется метод *Items.Add***

```
listBoxl.Items.Add("Limtu");
```

- **Для добавления несколько элементов используется метод *AddRange***

```
listBox1.Items.AddRange(new String[] {"Первый", "Второй", "Третий"});
```

- **С помощью метода *Items.Insert* можно добавлять элемент в отдельный индекс списка**

```
listBox1.Items.Insert(2, "Третьим будешь?");
```

- **С помощью метода *Items.Remove* можно удалить элемент из списка**

```
listbox1.Items.Remove(“string removed");
```

- **Метод *Items.RemoveAt* удаляет элемент под указанным индексом**

```
listBox1.Items.RemoveAt(2);
```

Что такое объект Graphics?

- **Объект Graphics :**
  - Предоставляет холст, на который выводится информация
  - Предоставляет методы для вывода текста и графики в указанную позицию
  - Предоставляет набор инструментов для изменения выводимой информации

```
Graphics myGraphic = this.CreateGraphics();

//draw lines or outlined shapes using a Pen
myGraphic.DrawLine(myPen,X1,Y1,X2,Y2) ;

//draw filled shapes using a Brush
myGraphic.FillRectangle(myBrush,X1,Y1,X2,Y2);

//draw text using a Font and a Brush
myGraphic.DrawString(myText,myFont,myBrush,X1,Y1);
```

Создание перьев, кистей и шрифтов

Перья

Перья используются для рисования линий и очертаний фигур

```
Pen myPen = new Pen(Color.Blue);
```

Кисти

Кисти используются для заливки фигур и текста

```
SolidBrush myBrush = new SolidBrush(Color.Blue);
```

Шрифты

Шрифты используются для вывода текста определенного размера и стиля

```
Font myFont = new Font("Arial", 12);
```

Рисование линий и фигур

- **Создайте объект *Graphics* вызовом метода**

System.Windows.Forms.Control.CreateGraphics.

- **Создайте объект *Pen.***
- **Вызовите член класса *Graphics* для рисования на элементе управления с помощью объекта *Pen.***

```
private void Form1_Paint(object sender, PaintEventArgs e)
    {
      Graphics g = this.CreateGraphics();
      Pen p = new Pen(Color.Red, 5);
      g.DrawRectangle(p, 10, 10, 100, 100);
    }
```

Настройка перьев

- **Для рисования пунктирной линии создайте экземпляр класса *Pen* и присвойте свойству *Pen.DashStyle* одно из следующих значений:**

  DashStyle.Dash, - mupe

  DashStyle.DashDot,

  DashStyle.DashDotDot,

  DashStyle.Dot - точка

  DashStyle. Solid.

  - используется пространство имен *System.Drawing.Drawing2D*

```
Graphics g = this.CreateGraphics();
        Pen p = new Pen(Color.Red, 5);
         p.DashStyle = DashStyle.Dot;
         g.DrawLine(p, 50, 25, 400, 25);
```

Заливка фигур

- **Применяются методы *Fill*.**
  - требуют экземпляр класса *Brush.*

    Класс *Brush* — абстрактный, поэтому сначала необходимо создать экземпляр одного из его производных классов

    System.Drawing.Drawing2D.HatchBrush

    System.Drawing. TextureBrush

    System.Drawing Drawing2D.PathGradientBrush

    System.Drawing.SolidBrush

    System.Drawing.Drawing2D.LinearGradientBrush

```
Graphics g = this.CreateGraphics();
Brush b = new SolidBrush(Color.Maroon);
// Создать массив точек points
g.FillPolygon(b, points)
```

Введение в многопоточное программирование

Многопоточность

Многопоточность — свойство платформы (например, операционной системы, виртуальной машины и т. д.) или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

Потоки выполнения (threads of execution, нити, треды, потоки, легковесные процессы) -
параллельно выполняющиеся
потоки управления в адресном пространстве
одного процесса.

Потоки выполнения и процессы

Потоки выполнения

- Различные потоки выполняются в одном адресном пространстве.
- Потоки выполнения имеют «собственный» стек и набор регистров. Глобальные данные являются общими.

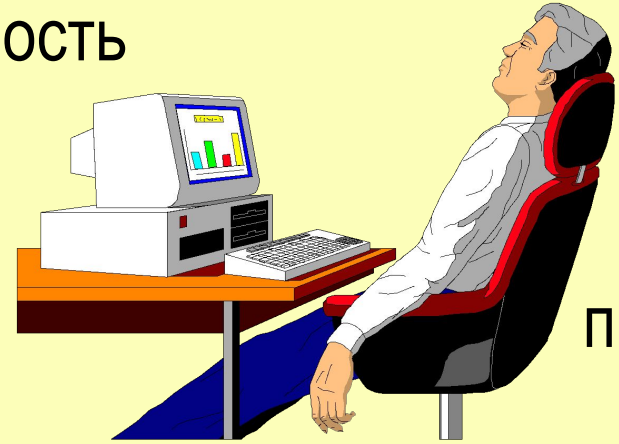
Процессы

- Различные процессы выполняются в разных адресных пространствах.
- Как локальные, так и глобал ьные переменные процессов являются «собственными».

Почему используются потоки выполнения ?

Производительность

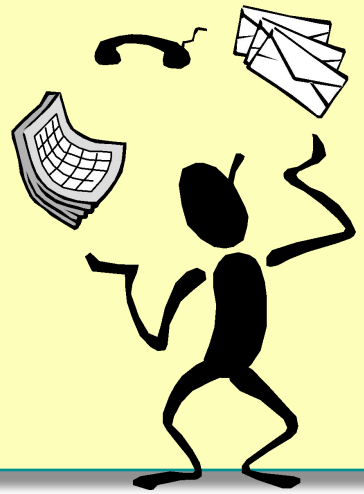
Восприятие производительности



Упрощение кодирования

Когда следует использовать потоки выполнения ?

- **Для использования технических возможностей многопроцессорных систем при сложных вычислениях;**
- **Отдельные потоки выполнения для различных задач:**
  - Сохранение пользовательского интерфейса «живым»;
  - Ожидание ввода/вывода;
  - Деление задач комплексного тестирования и циклов.
  - **Создание серверов**
  - Локальных и сетевых.



Проблемы при использовании потоков выполнения

- Разделяемые ресурсы;
- Взаимодействие между потоками выполнения;
- Чрезмерное использование многопоточности отнимает ресурсы и время CPU на создание потоков и переключение между потоками.

Синтаксис создания потоков выполнения

- **public delegate void ThreadStart()** – делегат, указывающий метод, который нужно выполнить.
- **Class Thread –** класс, используемый для создания потоков выполнения. Каждый экземпляр класса – отдельный поток выполнения.
- **Метод Start** (объявлен в классе Thread) - начинает выполнение потока. Поток продолжается до выхода из исполняемого метода либо при использовании метода **Interrupt или Abort**, которые прерывают выполнение потока.

```
using System.Threading;
...
ThreadStart  ts = new  ThreadStart(myMethod);
Thread  p1 = new  Thread(ts);
p1.Name = "MyThread";
p1.Start();
```

```
public void myMethod(){
        // Do some work
        ...
}
```