Объектно-ориентированное программирование

Лабораторные работы

Лабораторная работа 2

Задание :

Доработать класс (по заданию преподавателя) и программу, иллюстрирующую возможности данного класса

Интерфейс программы:

В программе должны быть:

- Несколько объектов данного класса,
- окна ввода параметров этих объектов,
- окна вывода вычисленных параметров и свойств объектов,
- кнопки, по которым вызываются функции, иллюстрирующие возможности этих объектов
- окно для графического отображения объектов.

Основная задача:

- Перегрузить заданные операции и показать их использование в программе
- Показать возможности графического отображения объектов

Common Language Runtime — CLR

Существуют два принципиально отличающихся вида приложений C++, которые можно разрабатывать в среде Visual C++:

- программы, написанные на версии языка C++, определенной стандартом ISO/IEC (International Standards Organization/International Electrotechnical Commision)

- программы, выполняющиеся под управлением среды CLR и реализованные с помощью расширенной версии языка C++, которая носит название C++/CLI

Общеязыковая исполняющая среда (Common Language Runtime) — это стандартизованная среда выполнения программ, написанных на широком диапазоне высокоуровневых языков, включая Visual Basic, C#, C++ и др.

C++ для среды CLR называется C++/CLI — это язык C++ для инфраструктуры CLI (Common Language Infrastructure)

Common Language Runtime — CLR

Общеязыковая исполняющая среда (Common Language Runtime) и набор библиотек, называемых библиотеками классов, образуют среду разработки Visual C++ , которая называется **.NET Framework**

Библиотека классов .NET Framework обеспечивает функциональную поддержку, которая необходима коду при выполнении под управлением среды CLR, независимо от применяемого языка программирования, поэтому программы, написанные на C++, C# или любом другом языке, поддерживающем среду .NET Framework, используют одни и те же библиотеки .NET

C++/CLI

Инфраструктура CLI — это спецификация среды виртуальной машины, которая позволяет приложениям, написанным на разнообразных высокоуровневых языках программирования, выполняться в различных системах без изменения и перекомпиляции оригинального исходного кода.

Инфраструктура CLI специфицирует стандарт промежуточного языка виртуальной машины, в который компилируется исходный код высокоуровневого языка программирования.

Инфраструктура CLI определяет общий набор **типов данных**, называемый общей системой типов (Common Type System — CTS), который должен использоваться программами, написанными на любом языке, ориентированном на реализацию инфраструктуры CLI. Можно определять собственные типы данных, но их определение должно подчиняться ряду правил, чтобы они были согласованы со спецификацией среды CLR.

Common Language Runtime — CLR

Код C++, выполняемый под управлением среды CLR, называется **управляемым кодом C++**, поскольку данные и код находятся под контролем среды CLR.

В программах CLR освобождение памяти, динамически выделенной для размещения данных, осуществляется автоматически, что позволяет исключить главный источник ошибок “родных” приложений C++.

В неуправляемом коде C++ необходимо самостоятельно заботиться о выделении и освобождении памяти во время выполнения программы, и самостоятельно обеспечивать безопасность.

Common Language Runtime — CLR

В среде CLR поддерживается собственный механизм динамической памяти.

Среда CLR автоматически очищает память, поэтому не нужно использовать оператор **delete** в программах, написанных для среды CLR.

Среда CLR может также время от времени упорядочивать память, избегая фрагментации.

Таким образом, среда CLR исключает вероятность утечек памяти и ее фрагментации. Механизм управления очисткой динамической памяти среды CLR называется сборкой “мусора”.

Для выделения памяти в программе C++/CLI используется оператор **gcnew** вместо оператора **new**, а префикс **gc** показывает, что память выделяется в **очищаемой динамической памяти**, а не в “родной куче” C++.

Указатели и ссылки в среде CLR

Применять обычные “родные” указатели C++ для очищаемой динамической памяти нельзя, так как местоположение данных изменяется из-за автоматической сборки “мусора” и исключения фрагментации неиспользуемых блоков.

Для доступа к объектам в динамической памяти, который позволяет обновлять адреса при перемещении элементов данных используются два способа: **отслеживаемые дескрипторы**, представляющие собой некоторые аналоги указателей из “родного” C++, и **отслеживаемые ссылки** — эквивалента “родных” ссылок C++ в программах CLR.

Common Language Runtime — CLR

Отслеживаемые дескрипторы (tracking handle) похожи на “родные” указатели C++, между ними есть и существенные отличия. Дескриптор хранит адрес, который автоматически обновляется сборщиком “мусора”, если объект, на который он ссылается, перемещается во время дефрагментации динамической памяти.

Но арифметика адресов, как у “родных” указателей, к таким указателям неприменима, кроме того, приведение дескрипторов не допускается.

Переменные, размещенные в динамической памяти среды CLR, включая ссылочные типы среды CLR, не могут быть объявлены в глобальной области памяти.

Отслеживаемый дескриптор

Пример:

Отслеживаемый дескриптор по имени **str**, который способен хранить адрес объекта типа **String**:

```
String^ str;
```

При объявлении дескриптора, он автоматически инициализируется нулем, поэтому ни на что не ссылается. Для явного обнуления дескриптора служит ключевое слово **nullptr**

```
Str = nullptr; // Обнулить дескриптор
```

Отслеживаемый дескриптор

Можно явно инициализировать дескриптор при его объявлении:

```
String^ str1(L"Hello!!!");
```

Этот оператор создает в динамической памяти объект **String**, который содержит строку, заключенную в скобки; адрес нового объекта помещается в переменную str1

`L"Hello!!!"` – тип const wchar_t*, а не String

Массивы среды CLR

Память для массива CLR выделяется в очищаемой динамической памяти. Тип переменной массива указан ключевым словом **array**. При этом нужно указать тип элементов массива в угловых скобках, следующих за ключевым словом **array**.

Объявление переменной массива (ссылки на массив типа int):

```
array<int>^ Arr;
```

Можно создать массив CLR, используя оператор **gcnew**, одновременно с объявлением переменной массива.

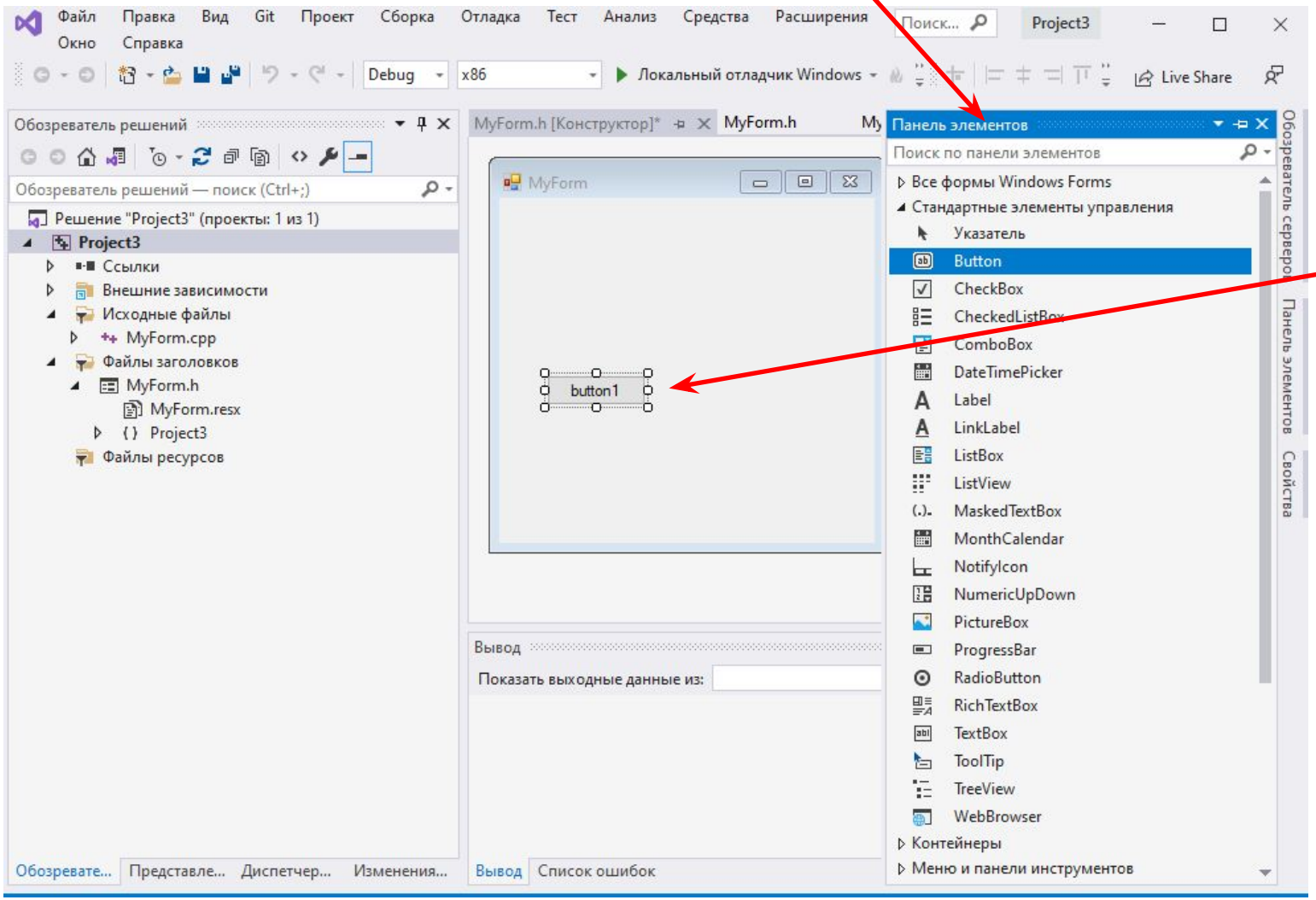
```
array<int>^ Arr = gcnew array<int>(100);
```

Создается массив для хранения 100 целых чисел

```
for(int i = 0 ; i < 100; i++)
  Arr[i] = i * 2;
```

Расположение компонент на форме

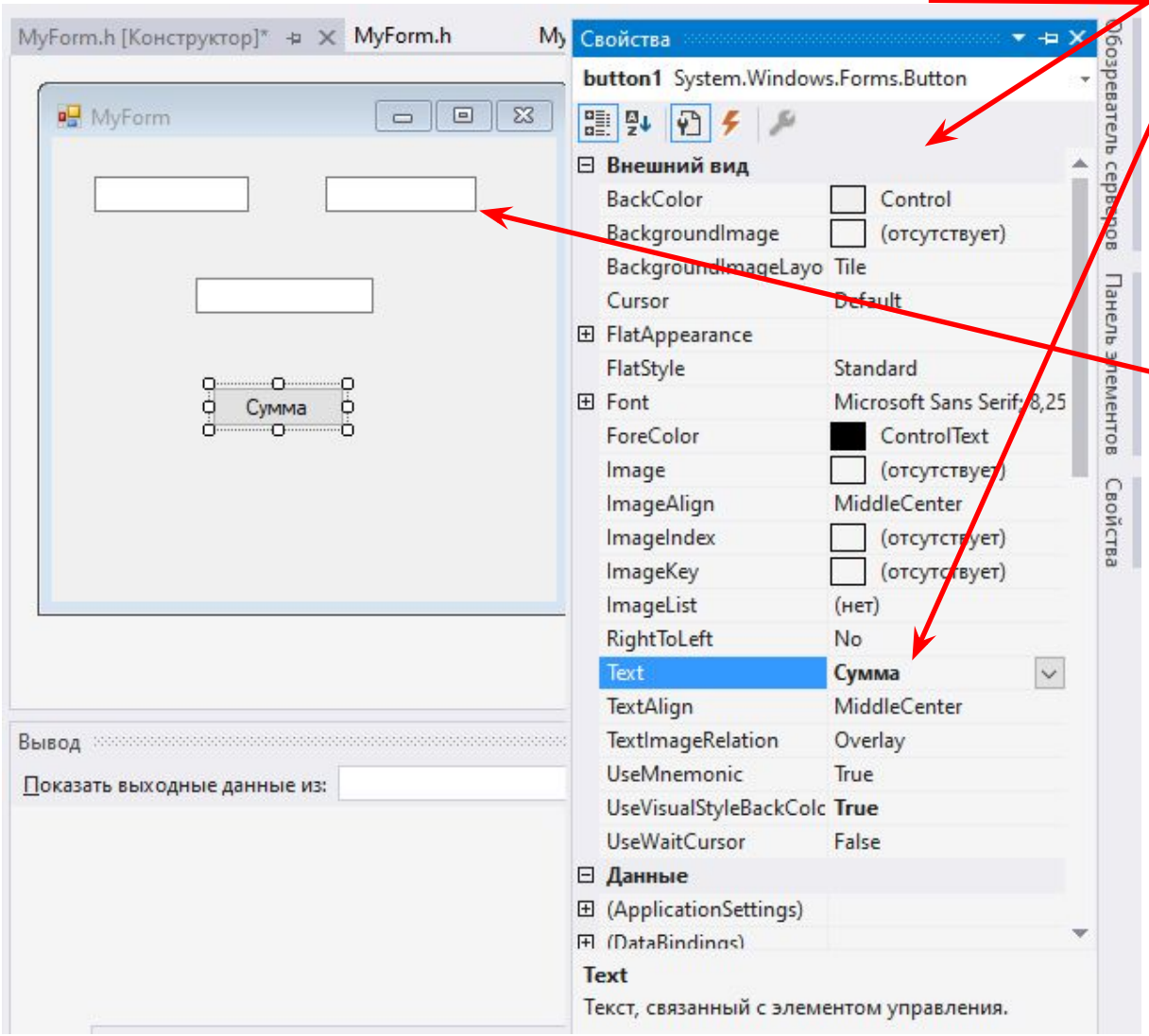
Компоненты берутся из ***Панели элементов*** *и размещаются на форме*



Расположение компонент на форме

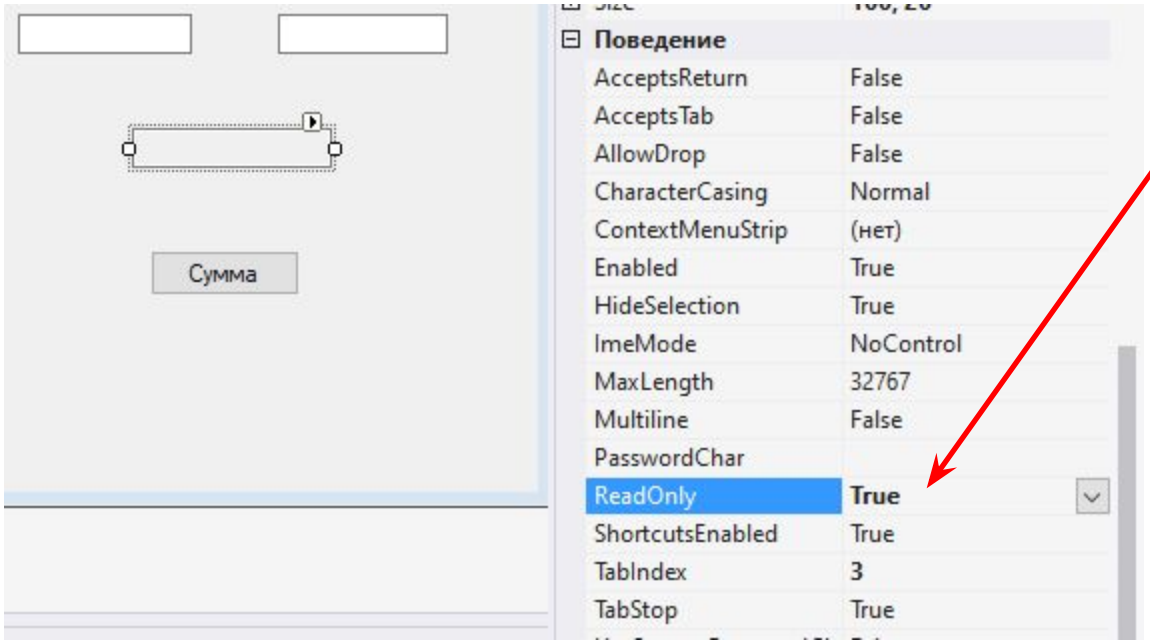
Свойства компонент задаются в *панели* ***Свойства***, например свойство ***Текст*** для компонента Button (кнопка)

Компоненты TextBox (окно ввода текста)



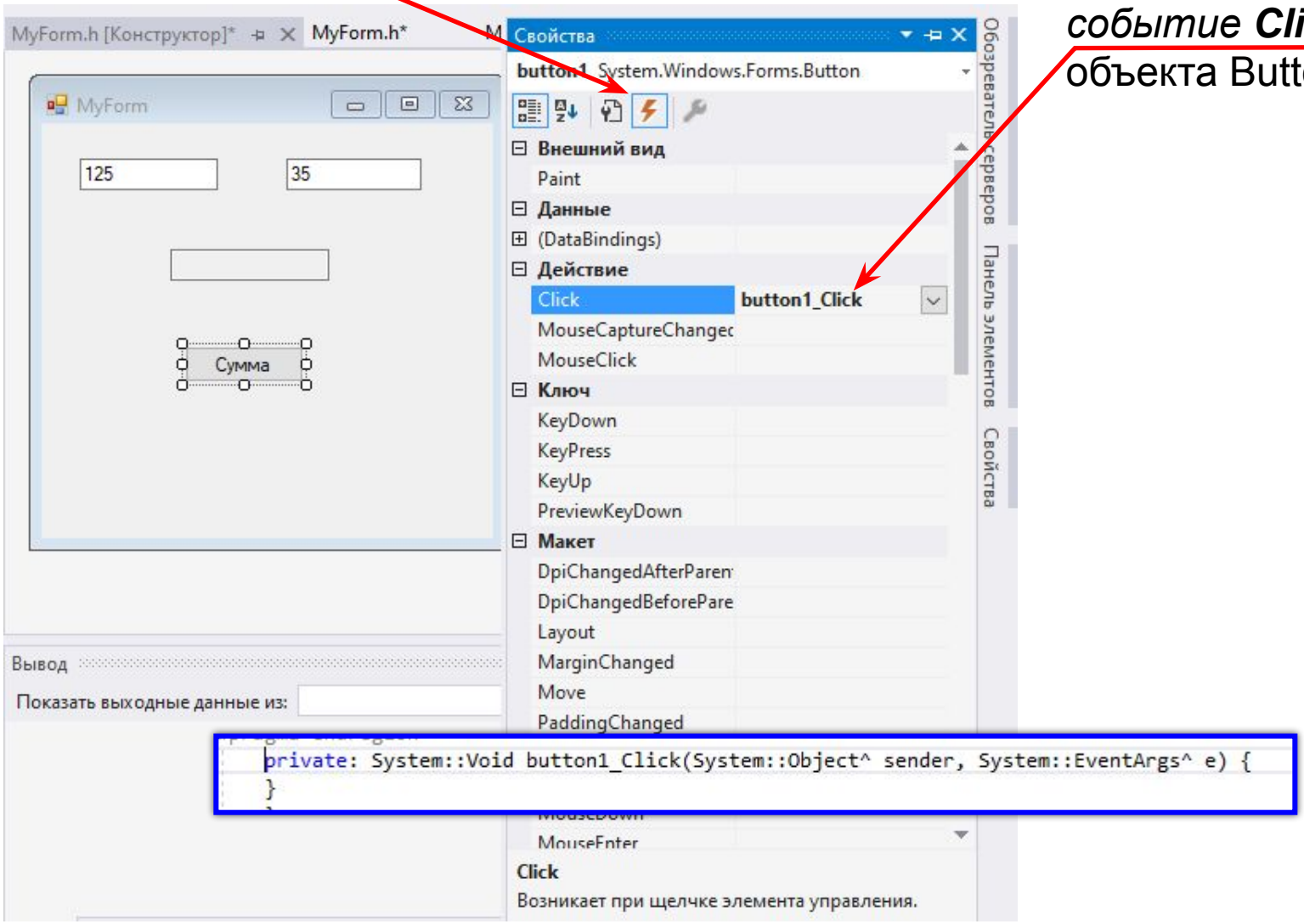
Расположение компонент на форме

Свойства компонент задаются в *панели* ***Свойства***, например свойство ***Только чтение*** для компонента TextBox



Создание событий для компонент

События для компонент задаются в *панели* ***Свойства***, например *событие* ***Click*** для объекта Button



Создание событий для компонент

```
System::Void button1_Click(System::Object^ sender,
                           System::EventArgs^ e)
{
  int x = Convert::ToInt32(textBox1->Text);
  int y = Convert::ToInt32(textBox2->Text); //
  int z = x + y;
  textBox3->Text = Convert::ToString(z);
}
```