#### Практическое занятие №11-12 (6 часов).

**«Создание простейших классов. Разработка оконного приложения»**

**Цель работы:** сформировать умения использования компонентов стандартных диалогов и системы меню, изучить основные свойства; сформировать умения по созданию процедур на основе событий компонентов, освоить алгоритм построения класса, освоить создание объектов класса.

**Оборудование, технические и программные средства:** персональный компьютер, среда программирования **Visual Studio**.

В результате выполнения практического занятия студент овладевает

следующими общими (ОК) и профессиональными компетенциями (ПК):

|  |  |
| --- | --- |
| **Код** | Наименование результата обучения |
| ОК 1. | Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам. |
| ОК 2. | Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности. |
| ОК 9. | Использовать информационные технологии в профессиональной деятельности. |
| ПК 1.1. | Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием. |
| ПК 1.2. | Разрабатывать программные модули в соответствии с техническим заданием. |
| ПК 1.3. | Выполнять отладку программных модулей с использованием специализированных программных средств. |
| ПК 1.4. | Выполнять тестирование программных модулей. |
| ПК 1.5. | Осуществлять рефакторинг и оптимизацию программного кода. |
| ПК 3.4. | Проводить сравнительный анализ программных продуктов и средств разработки, с целью выявления наилучшего решения согласно критериям, определенным техническим заданием. |

**Теоретическая часть.**

#### Все программы на языке C# создаются внутри классов. Структура класса имеет следующий вид:

#### Class Student

#### {

#### }

#### Класс – это абстрактный тип данных, т.е. это некоторый шаблон, на основе которого будут создаваться его экземпляры – объекты.

#### В Си-шарп классы объявляются с помощью ключевого слова class: [модификатор доступа] class [имя\_класса] {    //тело класса }

#### Тело класса может содержать поля, методы, свойства и события.

#### Поля описывают состояние, методы описывают поведение будущего объекта.

#### *Синтаксис класса:*

#### Class имя\_класса

#### {

#### Тип доступа тип имя переменной 1;// поле

#### Тип доступа тип имя переменной 2;

#### Тип доступа возвращаемый тип

#### Имя метода 1 (список параметров)//метод

#### {

#### Тело метода

#### }

#### },

#### Где тип доступа

#### - public – доступ к члену возможен из любого места одной сборки, либо из другой сборки, на которую есть ссылка; - protected – доступ к члену возможен только внутри класса, либо в классе-наследнике; - internal – доступ к члену возможен только из сборки, в которой он объявлен; - private – доступ к члену возможен только внутри класса; - protected internal - доступ к члену возможен из одной сборки, либо из класса-наследника другой сборки.

#### Пример класса:

#### namespace HelloWorld {   class Student // класс будет internal    {     public string name; //поле класса

#### public void vivod() // метод   {     Console.WriteLine(name);   }

#### }

#### Объекты содержат в себе статические поля и все методы. Экземпляры содержат нестатические поля.

#### Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса.

#### *Синтаксис создания объекта:*

#### Имя класса имя объекта = new имя класса().

#### При создании объекта класса происходит вызов конструктора класса.

## **Окно сообщения MessageBox**

Как правило, для вывода сообщений применяется элемент MessageBox. Однако кроме сообственно вывода строки сообщения данный элемент может устанавливать ряд настроек, которые определяют его поведение.

Для вывода сообщения в классе MessageBox предусмотрен метод **Show**, который имеет различные версии и может принимать ряд параметров. Рассмотрим одну из наиболее используемых версий:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public static DialogResult Show(      string text,      string caption,      MessageBoxButtons buttons,      MessageBoxIcon icon,      MessageBoxDefaultButton defaultButton,      MessageBoxOptions options  ) |

Здесь применяются следующие параметры:

text: текст сообщения

caption: текст заголовка окна сообщения

buttons: кнопки, используемые в окне сообщения. Принимает одно из значений перечисления **MessageBoxButtons**:

* AbortRetryIgnore: три кнопки Abort (Отмена), Retry (Повтор), Ignore (Пропустить)
* OK: одна кнопка OK
* OKCancel: две кнопки OK и Cancel (Отмена)
* RetryCancel: две кнопки Retry (Повтор) и Cancel (Отмена)
* YesNo: две кнопки Yes и No
* YesNoCancel: три кнопки Yes, No и Cancel (Отмена)

Таким образом, в зависимости от выбора окно сообщения может иметь от одной до трех кнопок.

icon: значок окна сообщения. Может принимать одно из следующих значений перечисления **MessageBoxIcon**:

* Asterisk, Information: значок, состоящий из буквы i в нижнем регистре, помещенной в кружок
* Error, Hand, Stop: значок, состоящий из белого знака "X" на круге красного цвета.
* Exclamation, Warning: значок, состоящий из восклицательного знака в желтом треугольнике
* Question: значок, состоящий из вопросительного знака на периметре круга
* None: значок у сообщения отсутствует

defaultButton: кнопка, на которую по умолчанию устанавливается фокус. Принимает одно из значений перечисления **MessageBoxDefaultButton**:

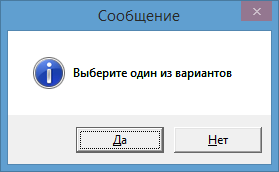
* Button1: первая кнопка из тех, которые задаются перечислением MessageBoxButtons
* Button2: вторая кнопка
* Button3: третья кнопка

options: параметры окна сообщения. Принимает одно из значений перечисления **MessageBoxOptions**:

* DefaultDesktopOnly: окно сообщения отображается на активном рабочем столе.
* RightAlign: текст окна сообщения выравнивается по правому краю
* RtlReading: все элементы окна располагаются в обратном порядке справа налево
* ServiceNotification: окно сообщения отображается на активном рабочем столе, даже если в системе не зарегистрирован ни один пользователь

Нередко используется один параметр - текст сообщения. Но посмотрим, как использовать остальные параметры. Пусть у нас есть кнопка, в обработчике нажатия которой открывается следующее окно сообщения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | private void button1\_Click(object sender, EventArgs e)  {      MessageBox.Show(          "Выберите один из вариантов",          "Сообщение",          MessageBoxButtons.YesNo,          MessageBoxIcon.Information,          MessageBoxDefaultButton.Button1,          MessageBoxOptions.DefaultDesktopOnly);  } |



Однако нам не просто дается возможность установки кнопок в окне сообщения. Метод MessageBox.Show возвращает объект **DialogResult**, с помощью которого мы можем узнать, какую кнопку в окне сообщения нажал пользователь. DialogResult представляет перечисление, в котором определены следующие значения:

* Abort: нажата кнопка Abort
* Retry: нажата кнопка Retry
* Ignore: нажата кнопка Ignore
* OK: нажата кнопка OK
* Cancel: нажата кнопка Cancel
* None: отсутствие результата
* Yes: нажата кнопка Yes и No
* No: нажата кнопка No

Используем обработку выбора пользователя, изменив обработчик нажатия кнопки следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | private void button1\_Click(object sender, EventArgs e)  {      DialogResult result = MessageBox.Show(          "Окрасить кнопку в красный цвет?",          "Сообщение",          MessageBoxButtons.YesNo,          MessageBoxIcon.Information,          MessageBoxDefaultButton.Button1,          MessageBoxOptions.DefaultDesktopOnly);        if (result == DialogResult.Yes)          button1.BackColor=Color.Red;        this.TopMost = true;  } |

И теперь, если в окне сообщения мы выберем выриант Yes, то кнопка окрасится в красный цвет.

#### Задание 1: Создайте простой класс Student.

#### Методические указания по выполнению задания:

* + 1. Запустите среду программирования **Visual Studio**. Создайте новое **Приложение Windows Forms**. Имя проекта и приложения **Student\_work**.
    2. Измените заголовок окна на **Работа с классом студент.**
    3. Измените свойство **Name** формы на **Student\_work.**
    4. Измените размеры окна формы (**Size)**.
    5. Добавьте в окно приложения элементы управления **MButton, TextBox, Label, numeric Up Down.** В свойстве Font установите размер шрифта равным **12 пт.** В свойстве **Text** задайте значения, соответствующие(из Панели элементов)

Фамилия имя

Рост

Рассчет веса - **кнопка**

* + 1. Добавьте класс Student в обозревателе решений с полями name, rost и ves (неизменяемое)
    2. Добавьте в класс Student метод SetEat, который будет отвечать за еду (ves увеличивается в зависимости от съеденного количества еды за минусом необходимых килокалорий).
    3. Для чтения данных из защищенного поля **ves** необходимо использовать еще один метод GetEat.

public double GetEat()

{

return ves;

}

* + 1. Создайте обработчик событий нажатием на кнопку Рассчитать вес, установив значение параметра eda сначала 1 кг, затем 5 кг, а потом проверьте его вес.

private void button1\_Click(object sender, EventArgs e)

{

Student stud = new Student();

stud.name = textBox1.Text;

stud.rost = (int)numericUpDown1.Value;

MessageBox.Show(string.Format("Студент: {0} \nрост: {1} \nвес: {2}", stud.name, stud.rost, stud.GetEat()));

………//рассчет веса, если студент съел 1 кг

………//рассчет веса, если студент съел 5 кг

}

* + 1. Запустите программу.
    2. Усовершенствуйте метод SetEat таким образом, что если студент за раз съедает более 5 кг, но менее 10 кг, то его рост уменьшается на 1 см, а вес увеличивается на 70 % от (eda\*1000 – 1600), виначе если студент съедает более 10 кг, то его рост уменьшается на 2 см, а вес увеличивается на 50 % от (eda\*1000 – 1800)

#### Запустите программу.

#### Задание 2: Создайте проект Windows Forms, в котором можно было бы вводить и получать информацию об определенных объектах некоторого класса (по вариантам, вариант – номер в журнале):

#### Задание 3: Создайте класс Робот с защитным полем, количеством жизней и методом поиска оставшегося количества жизней после игры. В зависимости от выпавшего случайного значения робот может потерять до половины своих жизней.

#### Методические указания по выполнению задания:

1. Запустите среду программирования **Visual Studio**. Создайте новое **Приложение Windows Forms**. Имя проекта и приложения **Robot**.
2. Измените заголовок окна в свойстве **Text** на **Игра робота.**
3. Измените свойство **Name** формы на **Robot.**
4. Измените размеры окна формы (**Size)**.
5. Добавьте в окно приложения элементы управления **Button, Label (2 шт).** В свойстве Font установите размер шрифта равным **12 пт.** В свойстве **Text** задайте значения, соответствующие(из Панели элементов)

Label1

Играть (Button)

Label2

1. Добавьте класс Robot в обозревателе решений

class robot

{

private int kollife;//закрытое поле, может использоваться только внутри класса

}

1. Для чтения данных из защищенного поля **kollife** необходимо использовать еще один метод **getlife**.

public int getlife()//метод для чтения из закрытого поля

{

return kollife;

}

1. Добавьте в класс Robot метод Min, который будет отвечать за сокращение количества жизней робота в два раза после игры. В качестве входного параметра а) будет задаваться значение текущего количества жизней робота:

public void min(int a)//метод, определяющий изменение количества жизней робота

{

kollife = a;

kollife = (int)kollife / 2;

}

1. Создайте обработчик событий нажатием на кнопку Играть. Количество жизней в игре у робота б) будет задаваться случайным образом:

private void button1\_Click(object sender, EventArgs e)

{

robot robot1 = new robot();

Random rn = new Random();

int b = rn.Next(0, 100);

label1.Text = "Количество жизней робота в начале игры " + Convert.ToString(b);

label2.Text = "Количество жизней робота после игры " + Convert.ToString(robot1.Getlife());

}

1. Запустите программу.
2. Измените текст программы так, чтобы поле kollife стало доступным и значение этого поля задавалось с клавиатуры.
3. Модифицируйте текст программы так, чтобы в системе существовало 3 робота с различным количеством жизней, задание которых осуществляется с клавиатуры.
4. Модифицируйте программу, добавив в нее метод, который позволит при уменьшении значения поля kollife конкретного робота на 50 % увеличить значение этого поля остальных роботов на 30 баллов, при уменьшении на 30 % - увеличить на 20 баллов.

namespace Robot

{

class robot

{

public int kollife;

public int getlife()

{

return kollife;

}

public void min (int a)

{

Random rn1 = new Random();

kollife = a;

kollife = rn1.Next(0,(a\*70/100));

}

public void kol (int a, int k)

{

if (k == a/2)

{

kollife = kollife +30;

}

else if (k == a\*0.7)

{

kollife += 20;

}

//else if (k>=a/2 && k<a\*0.7)

//{

// kollife += 40;

//}

//else if (k < a/2 && k>0)

//{

// kollife += 50;

//}

//else if (k == 0)

//{

// kollife += 100;

//}

}

}

}

private void button1\_Click(object sender, EventArgs e)

{

robot robot1 = new robot();

robot robot2 = new robot();

robot robot3 = new robot();

Random rn = new Random();

int b = rn.Next(0, 100);

robot1.min(b);

robot1.kollife = Convert.ToInt32(textBox1.Text);

label1.Text = "Количество жизней робота1 в начале игры "+ robot1.kollife;

robot2.kollife = Convert.ToInt32(textBox2.Text);

label6.Text = "Количество жизней робота2 в начале игры " + robot2.kollife;

robot3.kollife = Convert.ToInt32(textBox3.Text);

label7.Text = "Количество жизней робота3 в начале игры " + robot3.kollife;

int a = robot1.kollife;

robot1.min(robot1.kollife);

label2.Text = "Количество жизней робота1 в конце игры " + Convert.ToString(robot1.getlife());

robot2.kol(a, robot1.kollife);

label8.Text = "Количество жизней робота2 в конце игры " + Convert.ToString(robot2.getlife());

robot3.kol(a, robot1.kollife);

label9.Text = "Количество жизней робота3 в конце игры " + Convert.ToString(robot3.getlife());

}

}

}

**Задание 4**:Создайте новый проект для работы с приведенными в файле классами (самостоятельное задание 2 - по вариантам)

**Теоретические материалы.**

**Свойство** – это конструкция, которая заменяет методы доступа. Оно состоит из имени, типа и тела.

В теле задаются методы доступа с использованием ключевых слов **get** и **set**.

class Class1

{

public int vozrast; //Поле

public int Raschet // Свойства класса

{

get

{

return vozrast;

}

set

{

vozrast = value;

}

}

}

Метод **set** автоматически срабатывает, когда свойству присваивают значение. Метод **get** автоматически срабатывает, когда получают значение.

**Свойства позволяют реализовать инкапсуляцию.**

**Инкапсуляция** – это техника сокрытия частей класса.

Метод доступа **get** используется для получения значения из переменной.

Метод **set** используется для записи значения в переменную.

**Конструктор класса** – это специальный блок инструкций, вызываемый при создании объекта.

Создавать методы get и set необходимо для использования дополнительной логики.

При создании автоматически реализуемых свойств компилятор создаст закрытое, анонимное резервное поле, которое будет доступно с помощью методов get и set свойства.

Конструктор класса – специальный метод, который вызывается во время построения класса.

Если в теле класса не определен явно ни один конструктор, то всегда используется «невидимый» конструктор по умолчанию.

Имя конструктора всегда совпадает с именем класса.

Конструкторы не имеют возвращаемых значений.

Задача конструктора – инициализация полей значениями по умолчанию.

Задача пользовательского конструктора – инициализация полей предопределенными пользователем значениями.

**Задание 5**. 1. Измените программу задания 1, так чтобы вместо двух методов **SetEat** и **GetEat** появилось одно свойство **Ves**. Все имена полей напишите с маленькой буквы, а имя свойства с большой буквы.

Измените событие нажатием на кнопку.

1. Вернитесь к одному из ранее составленных проектов, работающих с классами (по вариантам) и измените классы так, чтобы в них появились свойства.
2. Сделайте отчет по практическому занятию.