Занятие 2. Основы ООП. Особенности ООП в языке программирования C#. *(ООП - объектно-ориентированное программирование)*

Введение.

Программирование, как наука, включает в себя намного больше, чем простое написание кода. Вообще, примерным временем рождения ***Computer Science*** (т.е. науки о вычислениях) можно считать конец Второй Мировой войны и послевоенное время. Как любые передовые технологии, компьютеры (а тогда всего лишь большие и громоздкие калькуляторы, размерами с Вашу комнату или даже больше) стали использоваться в первую очередь для военных целей и решения таких задач, как, например, взлом паролей. Основоположниками же «науки о вычислениях» были настоящие, фундаментальные ученые – математики, физики, инженеры. Лично для себя я разделяю такие понятия, как:

- ***кодинг*** - написание кода программы на каком-либо ЯП по заданной структуре, тех. заданию и т.п.; подразумевает хорошее знание синтаксиса и особенностей этого языка, опыт написания на этом языке;
- ***программирование*** – разработка, возможно только теоретическая, будущего приложения, подробное описание используемых алгоритмов, постановка целей и задач. Подразумевает знание некоторых теоретических основ, необходимых алгоритмов.

Так вот, к чему все это было сказано. А к тому, что не всегда наилучшим решением является сразу приступать к кодингу. Надо помнить о том, что хорошая программа – это не просто тонны бездумного кода, а целый комплекс, отлаженный механизм взаимодействия программы с пользователем или другими программами. В какой-то момент вам может понадобиться подправить что-то, что вы запрограммировали в начале, или что-то, что повлияет на многие игровые составляющие. Можно водрузить на нос очки и просматривать каждую строчку кода. Но куда удобнее, если у вас будет возможность поменять всего несколько строк в каком-то базовом (так называемом родительском) классе, а все зависящие от него (дочерние) изменятся соответственно. (Эти понятия подробнее рассмотрим ниже.) Следовательно, хорошим выбором будет реализовывать игру с оглядкой на принципы ООП. Ведь, не забываем, игра – это в конечном счете программа, и большинство фундаментальных идей программирования применимы и к созданию игр.

Объе́ктно-ориенти́рованное, или объектное, программи́рование (в дальнейшем ООП) - парадигма программирования, в которой основными концепциями являются понятия объектов и классов (Википедия). Не лучшее определение, кстати ☺ Данная парадигма, являющаяся самой распространенной и популярной в настоящее время, была придумана еще в 1967 году. Однако действительную популярность принцип приобрел благодаря языку C++, разработанному в 80-х.

Кстати, при объектном программировании имеет место быть и такой момент, что не обязательно знать, КАК это работает, чтобы нормально ИСПОЛЬЗОВАТЬ что-то в своей игре (хотя это и крайне желательно!). Т.е. можно переформулировать так – если вам что-то нужно, не спешите садиться за написание кода, а потратьте хотя бы полчаса-час на поиски готового решения в гугле. К примеру многие готовые решения и помощь в нахождении таковых можно найти по следующим адресам в интернете:

- http://create.msdn.com/en-US/education/catalog/ - официальный сборник примеров от Microsoft для XNA.
- http://www.codeplex.com/site/search?TagName=XNA&query=%22XNA%22 – сообщество программистов на .Net, в том числе много готовых решений для хна.
- http://www.cyberforum.ru/xna/ - русскоязычный форум программистов и сисадминов, один из лучших в рунете.

Определения основных понятий ООП.

Как говорилось выше, самые основные и главные понятия, постоянно используемые в ООП, это понятия ***объекта и класса.***

- ***Объект*** - это ограниченная сущность, характеризующаяся своим состоянием и поведением. Объект является ***экземпляром класса.***

  Т.е. говоря более простым языком, объект – это что-то такое, чему можно провести аналогию из реального мира, что-то конкретное и самостоятельное. К примеру, компьютер, за которым Вы сейчас сидите – это объект. Луна – объект. Из более сложных примеров можно сказать что-то типа «наше занятие – это объект».

- ***Класс*** – набор ***полей*** (или свойств) и ***методов*** (или функций), которые полностью характеризуют одну абстракцию (предмет, вещь, сущность). Этой абстракции не существует в реальности, т.е. во время выполнения программы. В реальности её можно получить только создав экземпляр данного класса, или, по-другому, ***объект.***
  Если обратиться к примерам из определения «объекта», то: компьютер в общем – это класс, спутник планеты – это класс, урок – это класс.

После этого, естественно, нам нужно понять, как же данные теоретические понятия реализовать в коде. Чтож, это несложно:

Краткий комментарий	Описание в коде
Все программы на языке C# представляют собой описание множества классов. Компьютеру известно ключевое слово «**class**», которое должно быть написано строчными буквами, но имя класса может быть любым и содержать как прописные, так и строчные буквы, причем использование пробелов не допускается и первой не должна стоять цифра.	**class MyClass** **{** **}**

Рассмотрим процесс объявления и создания объектов класса. Если планируется работать с определенным объектом, необходимо заранее сообщить компьютеру, к какому классу он принадлежит, для того чтобы компьютер проверил свои знания о нужном нам классе. Поэтому сначала указываем имя класса, а затем даем имя объекту. Этот процесс называется «объявлением объекта». Затем необходимо попросить компьютер «создать экземпляр» класса, что достигается с помощью ключевого слова «**new**». На основе своих знаний о классе компьютер создаст реальный объект, с которым он сможет выполнять действия. В нашем примере объект **myObject** становится экземпляром класса **MyClass**, или можно сказать, что **myObject** — это «объект» типа **MyClass**. Но в любом случае нам уже точно известно, что мы говорим об определенном объекте класса **MyClass**. Это называется «созданием объекта».	**MyClass myObject;** **myObject = new MyClass();** или **MyClass myObject = new MyClass();**

Другими, не менее важными понятими являются понятие ***поле и метод:***

- ***Поле*** – единичная характеристика объекта. Все поля ЭК характеризуют его текущее состояние. Вес, размер, цвет – все это простейшие примеры полей у объекта. Примерно тоже самое обозначает и термин *свойство,* но различия между ними сложно определить в несколько слов.
- ***Метод*** – набор команд для манипуляции с объектом, вынесенные отдельно, которые вызываются по *имени*. Можно так же определить, как «действие, которое будет выполнено ЭК».

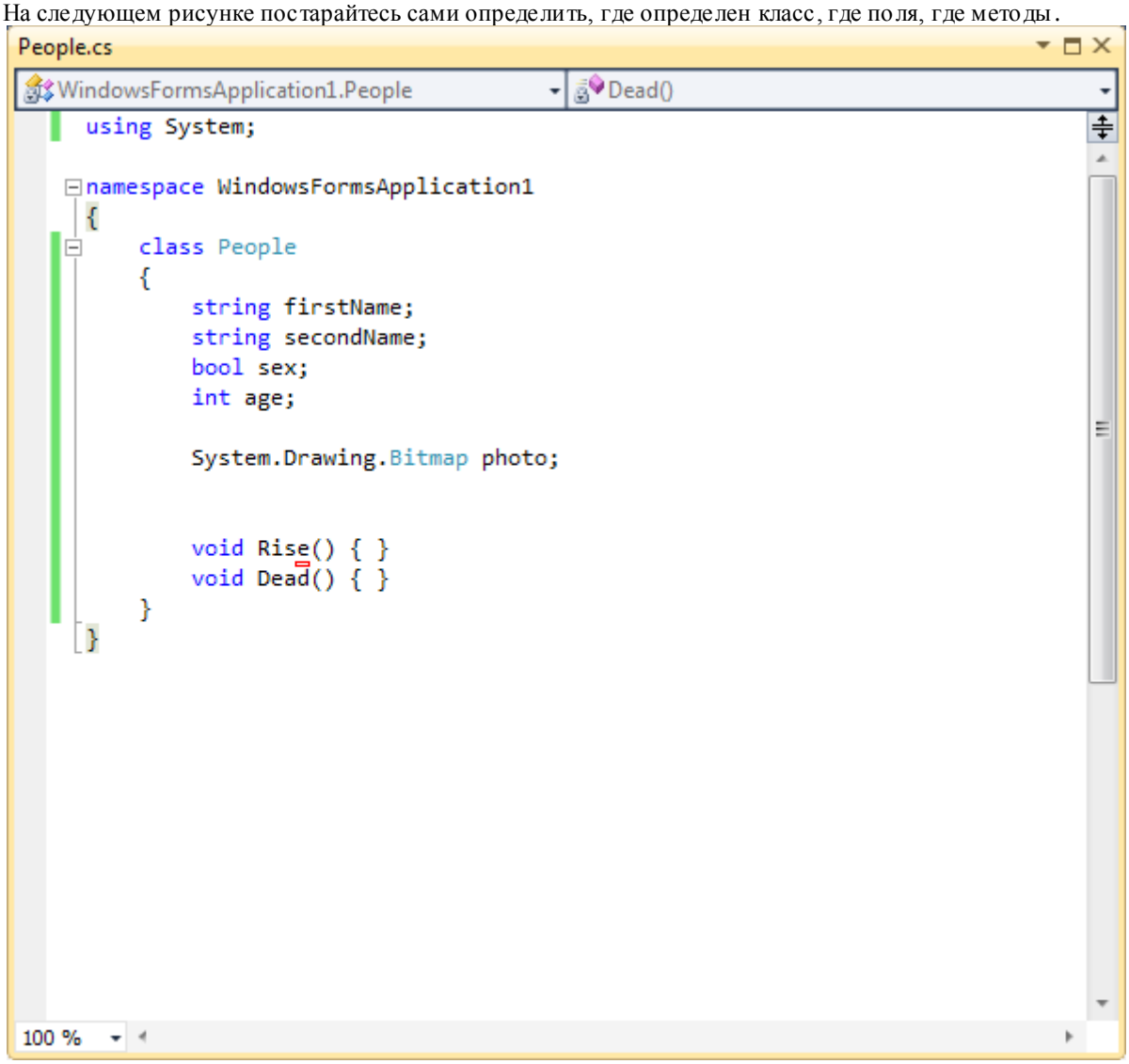
Реализация в коде:

Обычно в классе присутствует одно поле или несколько полей, которые позволяют описывать определенные черты объектов этого класса. Поля являются объектами, и описываются соответственно – тип поля плюс имя поля. Обращение к полю (т.е. получение значения, которое записано в этом поле) происходит путем написания имени объекта, точки, имени поля.	**class MyClass** **{** **int field1;** **string field2;** **float field3;** **bool field4;** **}** **MyClass myObject = new MyClass();** **myObject.field1;**

Метод – это просто набор команд. Соответственно, когда мы описываем логику метода, то должны руководствоваться всеми правилами, относящимся и к другим частям кода.	**class MyClass** **{** **void MyVoid()** **{**

Метод, как и поле, сначала нужно описать – это делается в теле класса, путем использования ключевого слова «**void**» и имени этого метода. Т.е. к примеру, void **MyVoid**. Соответственно вызвать команды, записанные в теле метода, можно так же, как и получить значение поля: путем написания имени объекта, точки, имени метода. И еще один момент. В круглых скобках можно перечислить так называемые параметры, т.е. переменные, которые «будут даны на вход» методу – это значит, что внутри метода мы сможем использовать эти объекты по имени, указанному в круглых скобках.	**}** **void MyVoid2(int i, string s)** **{** **}** **}** **MyClass myObject = new MyClass();** **myObject.MyVoid;**

На следующем рисунке постарайтесь сами определить, где определен класс, где поля, где методы.



Надеюсь Вы все правильно определили, так как пример то простой☺ Также стоит помнить, что хорошим тоном у программистов является привычка придерживаться подобного принципа - внутри тела класса (т.е. это все, что находиться между фигурными скобками) определение идет в следующем порядке:

- публичных полей
- потом приватных полей
- потом конструкторов
- потом деструкторов
- потом публичных методов
- потом приватных методов.

Данный порядок не является обязательным, т.е. если поменять что-то местами, то КОМПЬЮТЕР то Вас поймет. Однако, стоит заметить, что не только компьютер будет читать вашу программу *(это идеальный случай☺)*, периодически Вы сами будете проглядывать написанный ранее код, а возможно будете делиться

им с другими программистами. И вот тут то все зависит от Вас, будет ли товарищ-программиист проклинать Вас за «лапшу», без какой-либо структуры и логики, или же восхвалит весь небесный пантеон за такого аккуратного и прилежного молодого человека, что так красиво оформил свою программу. А понятия публичный и приватный рассматриваются дальше, не бойтесь.

Главные принципы ООП.

Объектно-ориентированное программирование, как таковое, характеризуется тремя простыми механизмами. Хотя каждый «пейсатель» книг по программированию (и ваш покорный слуга не исключение☺) стремиться дать свое определение для ООП, но эти принципы никто не оспаривает. Заключенные в них идеи фундаментальны и неизменны, хотя могут показаться достаточно банальными и невзрачными на первый взгляд. Итак, знакомьтесь: ***Инкапсуляция, Наследование, Полиморфизм.***

- ***Инкапсуляция*** - сокрытие данных и функций. Данные, определяющие состояние объекта, замкнуты в этом объекте.
  Что же это значит? А значит это то, что два объекта, даже являясь представителями одного класса, абсолютно независимы друг от друга. Вся информация, характеризующая данный объект – это его и только его забота. Другие объекты могут изменять свойства, вызывать методы – но они как была заключена внутри этого объекта, так и осталась.
- ***Наследование*** - это механизм, позволяющий строить иерархию классов. Некоторый класс объявляется как базовый (родительский), и из него порождается новый производный (дочерний) класс. Производный класс, обладая состоянием и поведением базового класса, дополняет их новыми, определёнными в нём самом, свойствами и функциями.
  Здесь просто приведу пример, банальный и заезженный до дыр, но очень и очень хороший и показательный. Кошка является дочерним классом класса млекопитающих, класс млекопитающих дочерним для класса животных. Т.е. понятно, что каждый следующий дочерний класс наследует все признаки родительского и расширяет, дополняет его своей собственной, доступной только ему функциональностью. Т.е. указание на теплокровность заложено в классе животных (если неправ – не судите строго, уроки биологии у меня были уже достаточно давно☺) и следовательно Кошка это свойство наследует.
- ***Полиморфизм*** - заключается в обозначении одного и того же действия одним именем, которое используется во всей иерархии порождения классов.
  Существует внешнее и внутреннее проявление полиморфизма. Внешнее достаточно сложно и требует введения еще нескольких новых, навороченных терминов и понятий. Да и не пригодиться оно нам в ближайшее время. Однако с понятием внутреннего стоит разобраться, так как и мы его будем использовать, и сам принцип мне кажется очень удачным и удобным. Суть механизма заключается в том, что методы класса могут иметь одинаковые имена, как это не подозрительно звучит. С одним лишь ограничением – они должны принимать на вход разный набор параметров или возвращать значения отличных типов.

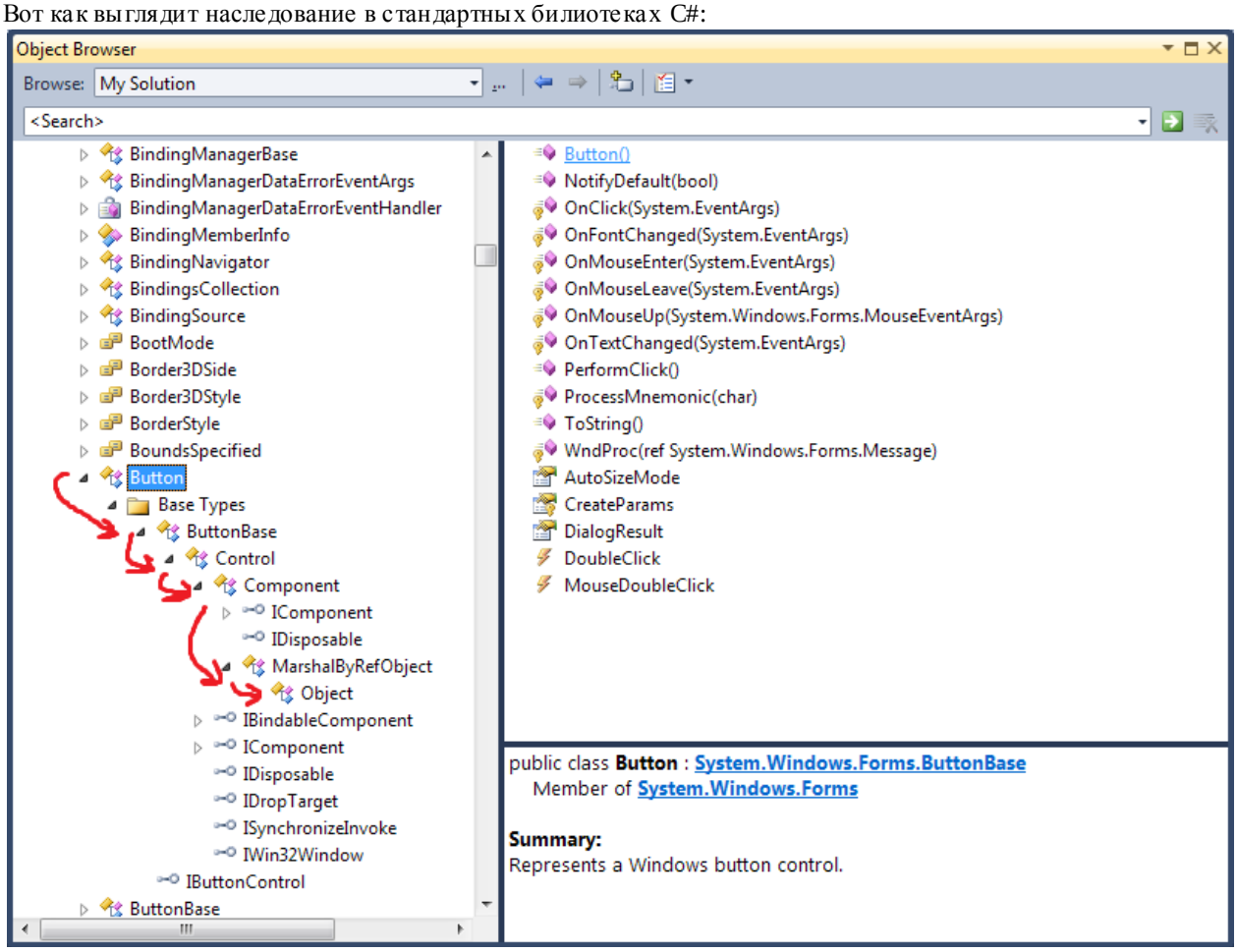
Как и раньше, покажу реализацию в коде:

Инкапсуляция, как таковая, в явном виде не обозначается – она просто пронизывает все-все объектное программирование.	

Механизм **наследования** нужно использовать с осторожностью. С одной стороны он позволяет избежать ненужных повторений кода, позволяет сделать структуру программы более логичной и обоснованной. Однако стоит избегать более чем трех уровней наследования, если это возможно, так как сложно будет держать это в голове в будущем. Это как в жизни – только самые ярые «гики» знают свою родословную больше трех-четырех колен, так как в повседневной жизни это слабо пригождается. А реализация достаточно простая – при объявлении нового класса после его имени ставится двоеточие «**:**» и указывается имя родительского класса.	**class ChildClass : ParentClass** **{** **}**
Стоит особенно выделить, что принцип **полиморфизма** применим только к методам. Т.е. класс и поля внутри класса должны, все-таки, иметь уникальные имена. Использование уже, наверное,	**class MyClass** **{** **void MyVoid()** **{**

понятно из определения – задание одного имени методу с разным набором параметров на вход.	`}`
	
	`void MyVoid(int i)`
	`{`
	`}`
	
	`void MyVoid(float f) { }`
	`}`

Вот как выглядит наследование в стандартных билиотеках C#:



Понятие доступности объекта.

Выше были использованы такие термины, как публичный и приватный. Они относятся к понятию доступности:

- ***Доступность объекта*** – это ограничение или наоборот предоставление прав доступа к объекту. Например, на получение значений его полей, изменение этих значений, возможность вызова методов объекта. Мы будем использовать 3 модификатора доступности: ***private, protected, public.***

Уточним значения этих терминов на примере полей (точно такое же значение они несут и для классов, методов):

- ***Private*** – «только сам объект может обращаться к данному полю».
- ***Protected*** – «только сам объект и объекты классов-наследников могут обращаться к полю»
- ***Public*** – «объекты любого класса могут обращаться к данному полю».
- По умолчанию (без явного указания) для методов, полей и классов стоит модификатор private.

Базовые типы данных в C#.

В C# существует чуть более десятка так называемых базовых типов данных. Т.е. это те типы данных (или классы, мы уже знает такой термин), без которых не может быть создана ни одна программа, те типы данных, аналоги или точные копии которых есть в любом мало-мальски серьезном языке программирования. Мы рассмотрим такие типы, как ***int, float, double, bool, string, char.***
Далее по списку в виде таблицы:

Название	Комментарий	Пример

INT	Целые числа.
	Размер: 8 бит.
	Диапазон значений:
	-2 147 483 648 * 2 147 483 647
		int i = 13;
		int i = -999;
		int i = 86 * 15;

FLOAT	Дробные числа, меньшая точность.
	Размер: 32 бита.
	Диапазон значений:
	-1,5 и еще 45 знаков * 3,4 и еще 38 знаков.
		float f = .01f;
		float f = 3.1415f;
		float f = -899.899f;

DOUBLE	Дробные числа, большая точность.
	Размер: 64 бита.
	Диапазон значений:
	-5 и еще 324 знака * 1,7 и еще 308 знаков.
		double d = 0.000001d;
		double d = 1d;
		double d = -123456789.123456789d;

BOOL	Логический тип данных.
	Размер: 1 бит.
	Диапазон значений:
	True, False.
		bool b = true;
		bool b = false;
		if (a != b) { } //в скобках тоже логическая переменная, только не заданная явно

STRING	Текстовый тип данных, текст. Размер: Диапазон значений:	string s = “f”; string s = “фух фух, Винни Пух”; string s = “12123 dfgdg 310 …dfg fgbl;kl ^^^45 lk,;l, dlk;nlknп варн зщоьзпздз дзпдрзтьд456456”;
CHAR	Одиночный символ. Размер: 16 бит. Диапазон значений: 0 * 65 535. Почему диапазон значений в виде чисел? Потому что переменная типа чар хранит в себе не сам символ, а его числовое представление в кодировке Unicode.	char c = ‘x’; char c = ‘1’; char c = ‘ɔ’;

Ух, это было сложное занятие. По крайне мере для меня☺ Но, очень надеюсь, что мне удалось донести основные принципы ООП до читателей, а более глубокие знания можно почерпнуть из книг профессиональных программистов/преподавателей, форумов и интернета.

Заключение.

Вот и завершился второй урок. Подведем итоги, с чем же мы ознакомились сегодня:

1. Что такое ООП.
2. С основными терминами, используемыми в парадигме ООП: класс, объект, поле, метод.
3. С основными механизмами, используемыми в парадигме ООП: инкапсулирование, наследование, полиморфизм.
4. С понятием доступности методов, полей и классов.
5. С некоторыми типами данных, которые принято называть базовыми.

Домашнее задание.

Постепенно мы приближаемся к тому, для чего и собрались здесь – программирование игр. Но сперва необходима интересная идея. Вот этим и займитесь дома: придумайте и запишите идею для своей

собственной игры. Можно взять классическую, но тогда нужна некая «фича», которой нет в других играх. Идея должна быть проста и эллегантна☺
Ну и попрактикуйтесь в программировании, используя полученные знания.