Товарищ полковник! Товарищи офицеры!

В соответствии планом сегодня мы проводим практическое занятие по дисциплине программирование на тему *«Фундаментальные концепции объектно-ориентированного программирования*»

Цели:

- углубить знания по объектно-ориентированному подходу в программировании, технологии программирования в интегрированной среде;

- изучить методику разработки, выполнения, отладки и тестирования простых программ на языке высокого уровня C#;

- получить понимание важности объектно-ориентированного программирования в профессиональной деятельности.

Учебные вопросы:

1. Основы наследования.
2. Конструкторы и наследование.
3. Абстрактные классы и сокрытие имен.

Важность рассматриваемой темы обусловлена необходимостью привития умения применять на практике одного из основных принципов ООП.

В этой связи хотелось бы напомнить, что наследование является одной из трех основных парадигм объектно-ориентированного программирования, наряду с полиморфизмом и инкапсуляцией. Кто нам может коротко раскрыть эти понятия? (вопрос к аудитории). Да, правильно.

1. Инкапсуляция - сокрытие реализации программных частей объекта.
2. Наследование - создание нового объекта/класса на основании уже имеющегося.
3. Полиморфизм – возможность использовать обобщённые методы, которые будут по-разному работать с разными типами данных.

На сегодняшнем занятии мы на практике столкнемся с уже рассматриваемыми на предыдущих лекционных и практических занятиях понятиями как классы, их члены, в том числе свойствами и конструкторами, ссылочные типы данных и другими.

Все материалы Вы как обычно можете получить, обратившись к электронной информационной обучающей среде нашей кафедры (нашему ftp-ресурсу).

Полученные в ходе данного курса и конкретного занятия навыки являются развитием ранее изучаемой Вами информатики и пригодятся в будущем при прохождении программирования СУБД.

Итак, давайте перейдем непосредственно к нашим учебным вопросам.

**Вопрос № 1. Основы наследования.**

У всех на рабочих местах запущена IDE-шка? Если, кто-то еще не успел, пожалуйста догоняйте.

Как я уже отмечал, наследование является одним из трех основополагающих принципов объектно-ориентированного программирования, поскольку оно допускает создание иерархических классификаций. Благодаря наследованию можно создать общий класс, в котором определяются характерные особенности, присущие множеству связанных элементов. От этого класса могут затем наследовать другие, более конкретные классы, добавляя в него свои индивидуальные особенности.

Поддержка наследования в C# состоит в том, что в объявление одного класса разрешается вводить другой класс. Для этого при объявлении производного класса (класса-наследника) указывается базовый класс. Рассмотрим для начала простой пример класса TwoDShape, содержащего ширину и высоту двухмерного объекта, например, квадрата, прямоугольника, треугольника и т.д. (проект 1.1)

В класс Triangle входят все члены его базового класса TwoDShape, и поэтому в нем переменные Width и Height доступны для метода Area(). Кроме того, объекты triangle 1 и triangle 2 в методе Main() могут обращаться непосредственно к переменным Width и Height, как будто они являются членами класса Triangle.

Несмотря на то что класс TwoDShape является базовым для класса Triangle, в то же время он представляет собой совершенно независимый и самодостаточный класс. Если класс служит базовым для производного класса, то это совсем не означает, что он не может быть использован самостоятельно.

В свою очередь, объект класса TwoDShape никак не связан с любым из классов, производных от класса TwoDShape, и вообще не имеет к ним доступа.

Для любого производного класса можно указать только один базовый класс.

В C# не предусмотрено наследование нескольких базовых классов в одном производном классе. Тем не менее можно создать иерархию наследования, в которой производный класс становится базовым для другого производного класса. Но в любом случае производный класс наследует все члены своего базового класса, в том числе переменные экземпляра, методы, свойства и индексаторы.

Главное преимущество наследования заключается в следующем: как только будет создан базовый класс, в котором определены общие для множества объектов атрибуты, он может быть использован для создания любого числа более конкретных производных классов. А в каждом производном классе может быть точно выстроена своя собственная классификация. В качестве примера ниже приведен еще один класс, производный от класса TwoDShape и инкапсулирующий прямоугольники (проект 1.1.1).

**Доступ к членам и наследование**

Члены класса зачастую объявляются закрытыми, чтобы исключить их несанкционированное или незаконное использование. Но наследование класса не отменяет ограничения, накладываемые на доступ к закрытым членам класса. Поэтому если в производный класс и входят все члены его базового класса, в нем все равно оказываются недоступными те члены базового класса, которые являются закрытыми. Так, если сделать закрытыми переменные класса TwoDShape, они станут недоступными в классе Triangle (проект 1.2).

Для преодоления ограничения на доступ к частным членам базового класса из производного класса в C# предусмотрены разные способы. Один из них состоит в использовании защищенных (protected) членов класса, а второй — в применении открытых свойств для доступа к закрытым данным.

Как пояснялось на предыдущих практических занятиях, свойство позволяет управлять доступом к переменной экземпляра. Например, с помощью свойства можно ввести ограничения на доступ к значению переменной или же сделать ее доступной только для чтения. Так, если сделать свойство открытым, но объявить его базовую переменную закрытой, то этим свойством можно будет воспользоваться в производном классе, но нельзя будет получить непосредственный доступ к его базовой закрытой переменной. (проект 1.3)

**Организация защищенного доступа**

Защищенный член класса создается с помощью модификатора доступа protected. Если член класса объявляется как protected, он становится закрытым, но за исключением одного случая, когда защищенный член наследуется. В этом случае защищенный член базового класса становится защищенным членом производного класса, а значит, доступным для производного класса. Таким образом, используя модификатор доступа protected, можно создать члены класса, являющиеся закрытыми для своего класса, но все же наследуемыми и доступными для производного класса. (проект 1.4.)

Модификатор доступа protected следует применять в том случае, если требуется создать член класса, доступный для всей иерархии классов, но для остального кода он должен быть закрытым. А для управления доступом к значению члена класса лучше воспользоваться свойством.

Итак, подведем итоги рассмотрения первого вопроса. Основными ошибками к сожалению, по-прежнему является невнимательность, пренебрежение к общепринятым подходам форматирования кода.

**Вопрос № 2. Конструкторы и наследование**

В иерархии классов допускается, чтобы у базовых и производных классов были свои собственные конструкторы. В связи с этим конструктор базового класса конструирует базовую часть объекта, а конструктор производного класса —производную часть этого объекта. И в этом есть своя логика, поскольку базовому классу неизвестны и недоступны любые элементы производного класса, а значит, их конструирование должно происходить раздельно. В приведенных выше примерах данный вопрос не возникал, поскольку они опирались на автоматическое создание конструкторов, используемых в C# по умолчанию. Но на практике конструкторы определяются в большинстве классов.

Если конструктор определен только в производном классе, то все происходит очень просто: конструируется объект производного класса, а базовая часть объекта автоматически конструируется его конструктором, используемым по умолчанию (проект 2.1).

Когда конструкторы определяются как в базовом, так и в производном классе, процесс построения объекта несколько усложняется, поскольку должны выполняться конструкторы обоих классов. В данном случае приходится обращаться к еще одному ключевому слову языка С# : base, которое находит двоякое применение: во-первых, для вызова конструктора базового класса; и во-вторых, для доступа к члену базового класса, скрывающегося за членом производного класса.

Для того чтобы продемонстрировать применение ключевого слова base на конкретном примере, рассмотрим еще один вариант класса TwoDShape (проект 2.2).

С помощью ключевого слова base можно вызвать конструктор любой формы, определяемой в базовом классе, причем выполняться будет лишь тот конструктор, параметры которого соответствуют переданным аргументам. В качестве примера в проекте 2.3 приведены расширенные варианты классов TwoDShape и Triangle, в которые включены как используемые по умолчанию конструкторы, так и конструкторы, принимающие один аргумент.

А теперь рассмотрим вкратце основные принципы действия ключевого слова base. Когда в производном классе указывается ключевое слово base, вызывается конструктор из его непосредственного базового класса. Следовательно, ключевое слово base всегда обращается к базовому классу, стоящему в иерархии непосредственно над вызывающим классом. Это справедливо даже для многоуровневой иерархии классов. Аргументы передаются базовому конструктору в качестве аргументов метода base(). Если же ключевое слово отсутствует, то автоматически вызывается конструктор, используемый в базовом классе по умолчанию.

На этом рассмотрение второго вопроса завершено. Мы увидели, как полезна на практике перегрузка конструкторов для возможности инициализации объектов класса с разными входными параметрами. Хотелось бы отметить работу курсантов Иванова и Петрова, которые активно участвовали в обсуждении вопроса.

**Вопрос № 3 Абстрактные классы и сокрытие имен**

В производном классе можно определить член с таким же именем, как и у члена его базового класса. В этом случае член базового класса скрывается в производном классе (проект 3.1).

**Применение ключевого слова base для доступа к скрытому имени**

Однако может случиться так, что Вам необходимо обратиться непосредственно к срываемому члену базового класса. Для этого существует ключевое слово base, которое действует по аналогии с ключевым словом this, но по отношению к классу-родителю нашего производного класса. В общем виде ключевое слово используется в форме: *base.член* где член может обозначать метод или переменную экземпляра. Эта форма ключевого слова base чаще всего применяется в тех случаях, когда под именами членов производного класса скрываются члены базового класса с теми же самыми именами (проект 3.2.).

С помощью ключевого слова base могут также вызываться скрытые методы (проект 3.3).

**Порядок вызова конструкторов многоуровневой реализации**

В иерархии классов конструкторы вызываются по порядку выведения классов: от базового к производному. Более того, этот порядок остается неизменным независимо от использования ключевого слова base. Так, если ключевое слово base не используется, то выполняется конструктор по умолчанию, т.е. конструктор без параметров (проект 3.4).

**Подходы к реализации абстрактных классов на основе виртуальных методов**

Давайте напомним себе немного теории. Виртуальным называется такой метод, который объявляется как virtual в базовом классе. Виртуальный метод отличается тем, что он может быть переопределен в одном или нескольких производных классах. Следовательно, у каждого производного класса может быть свой вариант виртуального метода. Кроме того, виртуальные методы интересны тем, что именно происходит при их вызове по ссылке на базовый класс. В этом случае средствами языка C# определяется именно тот вариант виртуального метода, который следует вызывать, исходя из типа объекта, к которому происходит обращение по ссылке, причем это делается во время выполнения.

Поэтому при ссылке на разные типы объектов выполняются разные варианты виртуального метода. Иными словами, вариант выполняемого виртуального метода выбирается по типу объекта, а не по типу ссылки на этот объект. Так, если базовый класс содержит виртуальный метод и от него получены производные классы, то при обращении к разным типам объектов по ссылке на базовый класс выполняются разные варианты этого виртуального метода.

Метод объявляется как виртуальный в базовом классе с помощью ключевого слова virtual, указываемого перед его именем. Когда же виртуальный метод переопределяется в производном классе, то для этого используется модификатор override. А сам процесс повторного определения виртуального метода в производном классе называется переопределением метода.

При переопределении имя, возвращаемый тип и сигнатура переопределяющего метода должны быть точно такими же, как и у того виртуального метода, который переопределяется. Кроме того, виртуальный метод не может быть объявлен как static или abstract.

Кажется, сложным? Давайте разбираться! Для более эффективной работы давайте разобьёмся на пары, что бы Вы могли подсказывать и перепроверять друг (проект 3.5)

**Применение виртуальных методов**

Для того чтобы стали понятнее преимущества виртуальных методов, применим их в классе TwoDShape. В предыдущих примерах в каждом классе, производном от класса TwoDShape, определялся метод Area(). Но, по-видимому, метод Area() лучше было бы сделать виртуальным в классе TwoDShape и тем самым предоставить возможность переопределить его в каждом производном классе с учетом особенностей расчета площади той двумерной формы, которую инкапсулирует этот класс. Именно это и сделано в приведенном ниже примере программы. Ради удобства демонстрации классов в этой программе введено также свойство Name в классе TwoDShape (проект 3.6)

Создавая собственные библиотеки классов, вы можете сами убедиться в том, что у метода зачастую отсутствует содержательное определение в контексте его базового класса. Подобная ситуация разрешается двумя способами. Один из них, как показано в предыдущем примере, состоит в том, чтобы просто выдать предупреждающее сообщение. Такой способ может пригодиться в определенных ситуациях, например, при отладке, но в практике программирования он обычно не применяется. Абстрактный метод создается с помощью указываемого модификатора типа abstract. У абстрактного метода отсутствует тело, и поэтому он не реализуется в базовом классе. Это означает, что он должен быть переопределен в производном классе, поскольку его вариант из базового класса просто непригоден для использования. Нетрудно догадаться, что абстрактный метод автоматически становится виртуальным и не требует указания модификатора virtual. В действительности совместное использование модификаторов virtual и abstract считается ошибкой. Для определения абстрактного метода служит приведенная ниже общая форма: abstract тип имя(список\_параметров);

Как видите, у абстрактного метода отсутствует тело. Модификатор abstract может применяться только в методах экземпляра, но не в статических методах (static). Абстрактными могут быть также индексаторы и свойства.

Класс, содержащий один или больше абстрактных методов, должен быть также объявлен как абстрактный, и для этого перед его объявлением class указывается модификатор abstract. А поскольку реализация абстрактного класса не определяется полностью, то у него не может быть объектов. Следовательно, попытка создать объект абстрактного класса с помощью оператора new приведет к ошибке во время компиляции.

Когда производный класс наследует абстрактный класс, в нем должны быть реализованы все абстрактные методы базового класса. В противном случае производный класс должен быть также определен как abstract. Таким образом, атрибут abstract наследуется до тех пор, пока не будет достигнута полная реализация класса.

Вот мы рассмотрели и третий вопрос. Типовыми ошибками являлись несоблюдения правил синтаксиса языка.

Итак, сегодня на занятии мы разобрали одну из самых важных и трудных для понимания тем. В этой связи необходимо дополнительно закрепить полученный материал в ходе самоподготовки.

**Задание на самостоятельную подготовку**

Изучить главу 8 литературы [3]. Стр. 172-186.

Лекция 8 литературы [2]. Cтр. 4 – 12

Решить задачу:

Создать абстрактный класс MilitaryVehicle (Военное транспортное средство). На его основе реализовать классы Plane (самолет), Саr (автомобиль) и Ship (корабль). Классы должны иметь возможность задавать и получать координаты и параметры средств передвижения (цена, скорость, год выпуска, боевые характеристики и т. п.) с помощью свойств. Для самолета должна быть определена высота, для самолета и корабля — количество пассажиров, для корабля — порт приписки. Динамические характеристики задать с помощью методов.

Разрешаю работать парами, как и в ходе изучения последнего вопроса.

Занятие окончено, спасибо за внимание!