# ПРАКТИЧЕСКАЯ РАБОТА №6

**Наследование классов**

**Цель работы**: научиться формировать классы-наследники, обращаться к открытым полям и методам класса-родителя, переопределять методы класса-родителя, обращаться к конструктору класса-родителя.

Выполнить две задачи объединённые в одну, используя методы

|  |  |  |
| --- | --- | --- |
| №  п/п | Ф.И.О. | НОМЕР ВАРИАНТА/ЗАДАНИЯ |
|  | Андриуца Дарья Николаевна | Набор примера из лекционного материала, 5 |
|  | Варфоломеев Иван Константинович | 1, 20 |
|  | Верховин Игорь Вадимович | 2,18 |
|  | Власов Тимур Михайлович | 3,19 |
|  | Гагауз Григорий Иванович | 4,17 |
|  | Голубцов Олег Русланович | Набор примера из лекционного материала, 3 |
|  | Гувирь Максим Григорьевич | Набор примера из лекционного материала, 4 |
|  | Дорожкина Алина Игоревна | 5, 16 |
|  | Жупанов Юлиан Игоревич | 6,15 |
|  | Капелько Иван Юрьевич | 4,14 |
|  | Кирста Андрей Александрович | 7,13 |
|  | Ковалев Иван Алексеевич | 8,12 |
|  | Лютов Глеб Васильевич | 9,11 |
|  | Михалаки Василий Федорович | 10,9 |
|  | Москалюк Иван Сергеевич | 11,8 |
|  | Огурцов Алексей Михайлович | 12,7 |
|  | Омельченко Евгений Борисович | 13,6 |
|  | Полякова Анна Игоревна | 14,5 |
|  | Редкозубов Владислав Романович | 15,4 |
|  | Ротарь Федор Федорович | 16,11 |
|  | Рошко (был Брага) Марин Михайлович | 17,12 |
|  | Салкуцан Анна Александровна | 18,13 |
|  | Скутельник Владимир Александрович | 19,14 |
|  | Струсевич Лев Дмитриевич | 20,15 |
|  | Телицкий Александр | 9,1 |
|  | Чернюк Егор Дмитриевич | 8,2 |
|  | Щитченко Николь Александровна | 7,3 |

# Теоретические сведения

Мощь ООП основана на **наследовании**. Когда построен полезный класс, то он может многократно использоваться. Повторное использование - это одна из главных целей ООП. Регулярно возникает необходимость для решения следующей задачи изменить разработанный ранее класс. Однако, делать это нельзя, так как класс уже используется в ряде программ и его изменение негативным образом скажется на их работе. В этом случае используют наследование класса. При этом создается новый класс, называемый классом-потомком, который объявляется наследником уже существующего класса.

Класс-**потомок** наследует от класса-родителя все поля и все методы, открытую и закрытую часть класса. При этом класс-потомок не получает доступ ко всем полям и методам класса-родителя. Поля и методы родительского класса, снабженные атрибутом *private*, наследуются, но остаются закрытыми, и методы, создаваемые классом-**потомком**, не могут к ним обращаться напрямую, а только через методы, у**наследованные** от класса-родителя. Класс-**потомок** – не наследует конструкторы родительского класса, их он должен создать сам.

Рассмотрим пример, в котором класс *Fnd*, играет роль класса-родителя. У него есть обычные поля, конструкторы и методы. Один из методов класса *Fnd* снабжен модификатором *virtual*, а другой - модификатором *override*.

Данные модификаторы ранее нами не использовались.

*public class Fnd*

*{*

*protected string name; protected int credit; public Fnd() { }*

*public Fnd(string name, int sum) { this.name = name; credit = sum; }*

*public virtual void VirtMethod() { Console.WriteLine ("Отец: " + this.ToString() ); }*

*public override string ToString() {*

*return(String.Format("поля: name = {0}, credit = {1}", name, credit)); } public void NonVirtMethod() {*

*Console.WriteLine ("Мать: " + this.ToString() ); } public void Analysis() {*

*Console.WriteLine ("Простой анализ"); } public void Work() {*

*VirtMethod(); NonVirtMethod(); Analysis(); }*

*}*

Здесь необходимо отметить, что класс *Fnd*, как и все классы, по

умолчанию является наследником класса *Object*, его потомки наследуют методы этого класса уже не напрямую, а через методы родителя, который мог переопределить методы класса *Object*. В частности, класс *Fnd* переопределил метод *ToString*(), задав собственную реализацию возвращаемой методом строки,которая связывается с объектами класса.

Модификатор метода *override* говорит о том, что класс-потомок переопределил данный метод, унаследованный от класса- родителя.

Класс *Fnd* закрыл свои поля для клиентов, но открыл для потомков, снабдив их модификатором доступа *protected*.

Создадим теперь класс *Dvd* - потомка класса *Fnd*. Добавим в класс- потомок новое поле класса, закрытое для клиентов этого класса, но открытое для его потомков: *protected int debet;*

Каждый класс должен позаботиться о создании собственных конструкторов. При создании конструкторов классов-потомков необходимо учитывать следующую особенность. Всякий конструктор создает объект класса - структуру, содержащую поля класса. Но конструктор потомка, прежде чем создать собственный объект, вызывает конструктор родителя, создавая родительский объект, который затем может быть дополнен полями потомка.

Вызов конструктора родителя происходит не в теле конструктора, а в заголовке, пока еще не создан объект класса. Для вызова конструктора используется ключевое слово *base*, именующее родительский класс.

*public Dv() {}*

*public Dv(string name, int cred, int deb):base (name,cred){ debet = deb;}*

Для конструктора без аргументов вызов аналогичного конструктора родителя подразумевается по умолчанию. Для конструкторов с аргументами вызов конструктора с аргументами родительского класса должен быть явным. Этот вызов синтаксически следует сразу за списком аргументов конструктора, будучи отделен от этого списка символом двоеточия.

Конструктору потомка передаются все аргументы, необходимые для инициализации полей, часть из которых передаются конструктору родителя для инициализации родительских полей.

Потомок может создать новый собственный метод с именем, отличным от имен наследуемых методов. В этом случае никаких особенностей нет.

*public void DvMethod(){ Console.WriteLine("Это метод класса Dvd");}*

В отличие от неизменяемых полей классов - родителей, класс - потомок может изменять наследуемые им методы. Если потомок создает метод с именем, совпадающим с именем метода предков, то возможны три ситуации:

* перегрузка метода. Она возникает, когда сигнатура создаваемого метода отличается от сигнатуры наследуемых методов предков. В этом слу- чае в классе потомка будет несколько перегруженных методов с одним име- нем;
* переопределение метода. Метод родителя в этом случае должен иметь модификатор *virtual* или *abstract*. При переопределении сохраняется сигнатура и модификаторы доступа наследуемого метода;
* скрытие метода. Если родительский метод не является виртуаль- ным или абстрактным, то потомок может создать новый метод с тем же име-

нем и той же сигнатурой, скрыв родительский метод в данном контексте. При вызове метода предпочтение будет отдаваться методу потомка, а имя на- следуемого метода будет скрыто. Скрытый родительский метод всегда может быть вызван, если при вызове метода перед его именем через точку указать ключевое слово *base*.

**Метод потомка, скрывающий метод родителя**, следует сопровождать модификатором *new*, указывающим на новый метод.

Класс *Fnd* имел в своем составе метод *Analysis*. Его потомок класс *Divd*

создает свой собственный метод анализа, скрывая метод родителя:

*new public void Analysis(){ base.Analysis(); Console.WriteLine("Сложный анализ");*

*}*

Если модификатор *new* опустить, он будет добавлен компилятором, при

этом компилятор выдаст сообщение - предупреждение о скрытии метода родителя. Здесь *потомок* строит свой анализ на основе метода, наследованного от родителя, вызывая первым делом скрытый родительский метод.

Рассмотрим случай, когда *потомок* добавляет перегруженный метод.

*public void Analysis(int level){ base.Analysis();*

*Console.WriteLine("Анализ глубины {0}", level);*

*}*

Большой ошибки не будет, если указать модификатор *new* и в этом случае, но будет выдано предупреждающее сообщение, что модификатор может быть опущен, поскольку сокрытия родительского метода не происходит.

**Статическим связыванием** называется связывание цели вызова и вызываемого метода на этапе компиляции, когда с сущностью связывается метод класса, заданного при объявлении сущности. Т.е. компилятор знает адреса методов, и организует обращение по этим адресам.

**Динамическим связыванием** называется связывание цели вызова и вызываемого метода на этапе выполнения, когда с сущностью связывается

метод класса объекта, связанного с сущностью в момент выполнения. Т.е. компилятор не знает адреса методов, они будут известны только в момент выполнения программы.

В языке C# принято следующее соглашение:

* по умолчанию предполагается статическое связывание;
* чтобы применить **динамическое связывание**, метод родительского класса должен снабжаться модификатором *virtual* или *abstract*, а его потомки должны иметь модификатор *override*.

Под **полиморфизмом** в ООП понимают способность одного и того же программного текста *x.M* выполняться по-разному, в зависимости от того, с каким объектом какого класса будет связан объект *x*. **Полиморфизм** гарантирует, что вызываемый метод *M* будет принадлежать классу объекта, связанному с сущностью *x*. В основе полиморфизма, характерного для семейства классов, лежат три механизма:

* одностороннее присваивание объектов внутри семейства классов; сущность, базовым классом которой является класс предка, можно связать с объектом любого из потомков. Другими словами, для введенной нами после- довательности объектов *xk* присваивание *xi = xj* допустимо для всех *j* >=*i*;
* переопределение потомком метода, наследованного от родителя. Благодаря переопределению, в семействе классов существует совокупность **полиморфных методов** с одним именем и сигнатурой;
* **динамическое связывание**, позволяющее в момент выполнения вызывать метод, который принадлежит целевому объекту.

В совокупности это и называется **полиморфизмом семейства классов**. Целевую сущность часто называют **полиморфной сущностью**, вызываемый метод - **полиморфным методом**, сам вызов - **полиморфным вызовом**.

Рассмотрим пример с классами *Fnd, Divd*. В родительском классе определен виртуальный метод *VirtMethod* и переопределен виртуальный метод *ToString*() родительского класса *object*. Потомок класса *Fnd* - класс *Dvd* переопределяет эти методы:

*public override void VirtMethod(){ Console.WriteLine("Сын: " + this.ToString());*

*}*

*public override string ToString(){*

*return(String.Format("поля: name = {0}, credit = {1},debet ={2}",name, credit, debet));*

*}*

В классе *Fnd* определены два невиртуальных метода *NonVirtmethod* и *Work*, наследуемый потомком *Dvd* без всяких переопределений. Рассмотрим более подробно работу метода *Work*:

*public void Work()*

*{*

*VirtMethod(); NonVirtMethod(); Analysis();*

*}*

При компиляции метода *Work* будет обнаружено, что вызываемый

метод *VirtMethod* является виртуальным, поэтому для него будет применяться **динамическое связывание**. Это означает, что вопрос об адресе метода откладывается до момента, когда метод *Work* будет вызван объектом, связанным с *x*. Объект может принадлежать как классу*Fnd*, так и классу *Dvd*, в зависимости от класса объекта и будет вызван метод этого класса.

Для не виртуальных методов *NonVirtMethod* и *Analysis* будет применено **статическое связывание**, так что *Work* всегда будет вызывать методы, принадлежащие классу *Found*. Здесь есть свои особенности: метод *NonVirtMethod*

*public void NonVirtMethod(){*

*Console.WriteLine ("Мать: "+ this.ToString());*

*}*

в процессе своей работы вызывает виртуальный метод *ToString*. Так как мы обратились к виртуальному методу, то для него будет использовано **динамическое связывание**, и в момент выполнения будет вызываться метод класса объекта.

Для вызова метода *Analysis*, определенного в каждом классе, при вызове метода *Work* будет организовано обращение только к родительскому методу из-за стратегии **статического связывания**.

В ряде случаев используют классы, для которых запрещается строить

классы-потомки путем наследования. Для этого достаточно приписать классу модификатор *sealed* - он запрещает построение потомков.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

Разработать иерархию классов. Выделить родительский класс. Описать конструкторы в каждом классе. Создать перегруженные и виртуальные методы, а также методы скрывающие методы родителя. Продемонстрировать работу объектов классов в приложении.

Перечень классов**:**

* 1. студент, преподаватель, персона;
  2. служащий, персона, рабочий;
  3. рабочий, кадры, администрация;
  4. деталь, механизм, изделие;
  5. страница, книга, печатное издание;
  6. тест, экзамен, испытание;
  7. местность, область, город;
  8. игрушка, продукт, товар;
  9. организация, цех, завод;
  10. персона, ребенок, ученик;
  11. автомобиль, двигатель, транспортное средство;
  12. страна, королевство, государство;
  13. млекопитающие, земноводные, животное;
  14. корабль, мачта, парусник;
  15. картина, произведение искусства, скульптура;
  16. музей, картинная галерея, выставка;
  17. книга, учебник, словарь;
  18. самолет, летательный аппарат, планер;
  19. маршрут, микроавтобус, водитель;
  20. компьютер, материнская плата, винчестер.