ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Утверждаю

Заместитель директора по УМР

\_\_\_\_\_\_\_\_\_\_\_\_/Е.А. Родзик

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Методические рекомендации по выполнению   
учебной-практической работы №3

учебной практики   
*УП 05. Программирование*

*Тема: «Отношение между классами»*

г. Томск – 2020 г

РАССМОТРЕННО

на заседании ПЦК

«Информационные системы и программирование»

протокол №\_\_\_\_\_\_

от «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Председатель ПЦК

\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Фунтиков М.Н.Рекомендации разработаны на основе Федерального государственного образовательного стандарта (далее – ФГОС) по специальностям среднего профессионального образования (далее СПО) 09.02.07 – «Информационные системы и программирование» и в соответствии с примерной основной образовательной программой.

Организация-разработчик:

ОГБОУ СПО «Томский техникум информационных технологий»

Разработчики:

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Сидиков И.Д.

\_\_\_\_\_\_\_\_\_\_\_\_\_/ Владимировна А. В.

**СОДЕРЖАНИЕ**

[1. Общая характеристика учебной-практической работы №3 4](#_Toc22557890)

[2. Краткие теоретические сведения 5](#_Toc22557891)

[3. Задания для выполнения 33](#_Toc22557892)

[4. Индивидуальные задания 34](#_Toc22557893)

[5. Контрольные вопросы 35](#_Toc22557894)

[6. Рекомендованная литература 36](#_Toc22557895)

[ПРИЛОЖЕНИЕ А 37](#_Toc22557896)

[ПРИЛОЖЕНИЕ Б 38](#_Toc22557897)

# ОБЩАЯ ХАРАКТЕРИСТИКА УЧЕБНОЙ-ПРАКТИЧЕСКОЙ РАБОТЫ №2

**Тема:** «Отношения между классами».

**Цель работы:** получение первоначальных навыков по работе и взаимоотношениями между классами и на языке программирования C#

**Проверяемые компетенции:**

ОК 1, ОК 2, ОК 4, ОК 9, ПК11.2.

**Инструкция по выполнению:**

1. Ознакомьтесь с теоретическим материалом, необходимым для выполнения практической работы.
2. Выполните предложенное практическое задание.
3. Оформите отчет по учебной практике (шаблон представлен в приложении)
4. Отправьте отчет на проверку.

**Время выполнения заданий:** 6 часов.

**Критерии оценки:**

|  |  |  |
| --- | --- | --- |
| **№** | **Критерий** | **Баллы** |
| 1 | Отчет соответствует предложенному шаблону | 1 |
| 2 | Задание выполнено правильно | 2 |
| 3 | В коде присутствует функции | 4 |
| 4 | Составлена блок схема к программе | 2 |
| 5 | Программа работает правильно без сбоев | 2 |
| 6 | Студент ответил на контрольные вопросы по лабораторной работе | 2 |
| 7 | Оформление отчета соответствует требованиям (шрифт, поля, отступы, интервалы, оформление рисунков, автоматическое оглавление) | 1 |
| 8 | Своевременность выполнения задания | 1 |
| Итого | | 15 |

**Перевод в пятибалльную систему оценивания:**

«отлично» - 13 - 15 баллов

«хорошо» - 10 -12 баллов

«удовлетворительно» - 7-9

«неудовлетворительно» <7 баллов

# КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## **Введение классы и структуры**

Классы и структуры являются двумя основными конструкциями системы общих типов CTS, используемой в платформе .NET Framework. Оба они являются структурами данных, которые инкапсулируют набор данных и поведений в одной логической сущности. Данные и поведение являются членами класса или структуры. К ним относятся методы, свойства, события и другие элементы, которые описаны далее в этой статье.

Объявление класса или структуры служит своего рода чертежом для создания экземпляров или объектов во время выполнения. Если вы определите класс или структур с именем Person, то Person здесь обозначает имя типа. Если вы объявите и инициализируете переменную p типа Person, принято говорить, что p является объектом (или экземпляром) Person. Можно создать несколько экземпляров одного типа Person, и каждый экземпляр будет иметь разные значения свойств и полей.

Класс является ссылочным типом. Когда вы создаете объект класса и сохраняете его в переменную, эта переменная содержит только ссылку на память объекта. Если ссылка на объект сохраняется в новую переменную, эта переменная также ссылается на исходный объект. Изменения, внесенные через одну переменную, отражаются и в другой переменной, поскольку обе они ссылаются на одни и те же данные.

Структура (struct) является типом значения. При создании структуры переменная, которой присвоена структура, содержит фактические данные этой структуры. Если структура присваивается новой переменной, все данные копируются. Таким образом, новая переменная и исходная переменная содержат две отдельные копии одинаковых данных. Изменения, внесенные в одну копию, не влияют на другую.

В общем случае классы используются для моделирования более сложного поведения или для таких данных, которые будут изменяться после создания объекта класса. Структуры лучше подходят для небольших структур данных, информация в которых не должна изменяться после создания структуры.

**Пример**

В следующем примере в пространстве имен ProgrammingGuide определен класс CustomClass с тремя членами: конструктор экземпляра, свойство с именем Number и метод с именем Multiply. Метод Main в классе Program создает экземпляр (объект) класса CustomClass. Обращение к методам и свойствам объекта осуществляется с использованием точечной нотации.

|  |
| --- |
| С# |
| using System;  namespace ProgrammingGuide  {  // Class definition.  public class CustomClass  {  // Class members.  //  // Property.  public int Number { get; set; }  // Method.  public int Multiply(int num)  {  return num \* Number;  }  // Instance Constructor.  public CustomClass()  {  Number = 0;  }  }  // Another class definition that contains Main, the program entry point.  class Program  {  static void Main(string[] args)  {  // Create an object of type CustomClass.  CustomClass custClass = new CustomClass();  // Set the value of the public property.  custClass.Number = 27;  // Call the public method.  int result = custClass.Multiply(4);  Console.WriteLine($"The result is {result}.");  }  }  }  // The example displays the following output:  // The result is 108. |

## **Общая теория классы**

### Ссылочный тип

Тип, который определен как класс, является ссылочным типом. Когда во время выполнения вы объявляете переменную ссылочного типа, такая переменная будет содержать значение NULL, пока вы явным образом не создадите экземпляр класса с помощью оператора new или не назначите его объекту совместимого типа, созданному в другом месте, как показано в следующем примере:

|  |
| --- |
| С# |
| //Declaring an object of type MyClass.  MyClass mc = new MyClass();  //Declaring another object of the same type, assigning it the value of the first object.  MyClass mc2 = mc; |

При создании объекта выделяется достаточный объем памяти для этого объекта в управляемой куче, и переменная хранит только ссылку на расположение данного объекта. Хранение типов в управляемой куче требует дополнительных действий как при выделении памяти, так и при удалении, которое выполняется функцией автоматического управления памятью в среде CLR, известной как сборка мусора. Сборка мусора является хорошо оптимизированным процессом и в большинстве случаев не создает помех для производительности.

### Объявление классов

Классы объявляются с помощью ключевого слова class, за которым следует уникальный идентификатор, как показано в следующем примере:

|  |
| --- |
| С# |
| //[access modifier] - [class] - [identifier]  public class Customer  {  // Fields, properties, methods and events go here...  } |

Ключевому слову class предшествует уровень доступа. Так как в этом случае используется открытый класс (public), любой пользователь может создавать его экземпляры. За именем класса следует ключевое слово class. Имя класса должно быть допустимым именем идентификатора C#. Оставшаяся часть определения — это тело класса, в котором задаются данные и поведение. Поля, свойства, методы и события в классе собирательно называются членами класса.

### Создание объектов

Несмотря на то, что они иногда взаимозаменяемы, класс и объект — разные вещи. Класс определяет тип объекта, но не является объектом. Объект — это конкретная сущность, основанная на классе, которую иногда называют экземпляром класса.

Объекты можно создавать с помощью ключевого слова new, за которым следует имя класса, на котором будет основан объект, например следующим образом:

|  |
| --- |
| С# |
| Customer object1 = new Customer(); |

При создании экземпляра класса ссылка на объект передается программисту. В предыдущем примере object1 представляет собой ссылку на объект, который основан на Customer. Эта ссылка указывает на новый объект, но не содержит данные этого объекта. Фактически, можно создать ссылку на объект без создания собственно объекта:

|  |
| --- |
| С# |
| Customer object2; |

Создание таких ссылок, которые не указывают на объект, не рекомендуется, так как попытка доступа к объекту по такой ссылке приведет к сбою во время выполнения. Однако такую ссылку можно сделать указывающей на объект, создав новый объект или назначив ее существующему объекту, как показано далее:

|  |
| --- |
| С# |
| Customer object3 = new Customer();  Customer object4 = object3; |

В этом коде создаются две ссылки на объект, которые указывают на один и тот же объект. Таким образом, любые изменения объекта, выполненные посредством object3, отражаются при последующем использовании object4. Поскольку на объекты, основанные на классах, указывают ссылки, классы называют ссылочными типами.

### Наследование классов

Классы полностью поддерживают наследование, фундаментальный механизм объектно ориентированного программирования. Создаваемый класс может наследовать от любого другого интерфейса или класса, который не определен как запечатанный, а другие классы могут наследовать от этого класса и переопределять его виртуальные методы.

При наследовании создается производный класс, то есть класс объявляется с помощью базового класса, от которого он наследует данные и поведение. Базовый класс задается добавлением после имени производного класса двоеточия и имени базового класса, как показано далее:

|  |
| --- |
| С# |
| public class Manager : Employee  {  // Employee fields, properties, methods and events are inherited  // New Manager fields, properties, methods and events go here...  } |

Когда класс объявляет базовый класс, он наследует все члены базового класса, за исключением конструкторов.

В отличие от C++, класс в C# может только напрямую наследовать от одного базового класса. Тем не менее, поскольку базовый класс может сам наследовать от другого класса, класс может косвенно наследовать от нескольких базовых классов. Кроме того, класс может напрямую реализовать несколько интерфейсов.

Класс может быть объявлен абстрактным. Абстрактный класс содержит абстрактные методы, которые имеют определение сигнатуры, но не имеют реализации. Нельзя создавать экземпляры абстрактных классов. Они могут использоваться только через производные классы, реализующие абстрактные методы. И наоборот, запечатанный класс не позволяет другим классам быть от него производными.

### Пример

В следующем примере определяется открытый класс, содержащий автоматически реализуемое свойство, метод и специальный метод, который называется конструктором. тем создаются экземпляры этого класса с помощью ключевого слова new.

|  |
| --- |
| С# |
| using System;  public class Person  {  // Constructor that takes no arguments:  public Person()  {  Name = "unknown";  }  // Constructor that takes one argument:  public Person(string name)  {  Name = name;  }  // Auto-implemented readonly property:  public string Name { get; }  // Method that overrides the base class (System.Object) implementation.  public override string ToString()  {  return Name;  }  }  class TestPerson  {  static void Main()  {  // Call the constructor that has no parameters.  var person1 = new Person();  Console.WriteLine(person1.Name);  // Call the constructor that has one parameter.  var person2 = new Person("Sarah Jones");  Console.WriteLine(person2.Name);  // Get the string representation of the person2 instance.  Console.WriteLine(person2);  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  // Output:  // unknown  // Sarah Jones  // Sarah Jones |

### Объекты

Определение класса или структуры подобно чертежу, на котором указаны действия, выполняемые типом. В сущности, объект является блоком памяти, выделенной и настроенной в соответствии с чертежом. Программа может создать множество объектов одного класса. Объекты также называют экземплярами. Они могут храниться либо в именованной переменной, либо в массиве или коллекции. Клиентский код — это код, использующий эти переменные для вызова методов и доступа к открытым свойствам объекта. В объектно-ориентированном языке, таком как C#, стандартная программа состоит из нескольких динамически взаимодействующих объектов.

### Экземпляры структуры и Экземпляры классов

Так как классы являются ссылочными типами, в переменной объекта класса хранится ссылка на адрес объекта в управляемой куче. Если первому объекту назначен второй объект того же типа, обе переменные ссылаются на объект, расположенный по данному адресу. Эта особенность обсуждается более подробно далее в этом разделе.

Экземпляры классов создаются с помощью оператора new. В приведенном ниже примере Person является типом, а person1 и person 2 — экземплярами или объектами этого типа.

|  |
| --- |
| С# |
| public class Person  {  public string Name { get; set; }  public int Age { get; set; }  public Person(string name, int age)  {  Name = name;  Age = age;  }  // Other properties, methods, events...  }  class Program  {  static void Main()  {  Person person1 = new Person("Leopold", 6);  Console.WriteLine("person1 Name = {0} Age = {1}", person1.Name, person1.Age);  // Declare new person, assign person1 to it.  Person person2 = person1;  // Change the name of person2, and person1 also changes.  person2.Name = "Molly";  person2.Age = 16;  Console.WriteLine("person2 Name = {0} Age = {1}", person2.Name, person2.Age);  Console.WriteLine("person1 Name = {0} Age = {1}", person1.Name, person1.Age);  // Keep the console open in debug mode.  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  /\*  Output:  person1 Name = Leopold Age = 6  person2 Name = Molly Age = 16  person1 Name = Molly Age = 16  \*/ |

Так как структуры являются типами значений, в переменной объекта структуры хранится копия всего объекта. Экземпляры структур также можно создавать с помощью оператора new, однако он не является обязательным, как показано в следующем примере:

|  |
| --- |
| С# |
| public struct Person  {  public string Name;  public int Age;  public Person(string name, int age)  {  Name = name;  Age = age;  }  }  public class Application  {  static void Main()  {  // Create struct instance and initialize by using "new".  // Memory is allocated on thread stack.  Person p1 = new Person("Alex", 9);  Console.WriteLine("p1 Name = {0} Age = {1}", p1.Name, p1.Age);  // Create new struct object. Note that struct can be initialized  // without using "new".  Person p2 = p1;  // Assign values to p2 members.  p2.Name = "Spencer";  p2.Age = 7;  Console.WriteLine("p2 Name = {0} Age = {1}", p2.Name, p2.Age);  // p1 values remain unchanged because p2 is copy.  Console.WriteLine("p1 Name = {0} Age = {1}", p1.Name, p1.Age);  // Keep the console open in debug mode.  Console.WriteLine("Press any key to exit.");  Console.ReadKey();  }  }  /\*  Output:  p1 Name = Alex Age = 9  p2 Name = Spencer Age = 7  p1 Name = Alex Age = 9  \*/ |

Память для p1 и p2 выделена в стеке потока. Эта память освобождается вместе с типом или методом, в котором она объявляется. Эта одна из причин того, почему структуры копируются при присваивании. Напротив, при выходе всех ссылок на объект из области действия среда CLR автоматически освобождает память (выполняет сборку мусора), выделенную для экземпляра класса. Возможность детерминированного уничтожения объекта класса, имеющаяся в C++, в данном случае отсутствует.

### Идентификация объектов и равенство значений

Сравнивая два объекта на предмет равенства, сначала необходимо определить, нужно ли узнать, представляют ли две переменные один объект в памяти или значения одного или нескольких их полей являются равными. Если вы планируете сравнить значения, следует решить, являются ли объекты экземплярами типов значений (структурами) или ссылочными типами (классами, делегатами, массивами).

* Чтобы определить, ссылаются ли два экземпляра класса на одно расположение в памяти (то есть имеют одинаковый идентификатор), воспользуйтесь статическим методом Equals. (System.Object является неявным базовым классом для всех типов значений и ссылочных типов, включая структуры и классы, определенные пользователем.)
* Чтобы определить, имеют ли поля экземпляра в двух экземплярах структуры одинаковые значения, воспользуйтесь методом ValueType.Equals. Так как все структуры неявно наследуются от System.ValueType, метод можно вызвать непосредственно в объекте, как показано в следующем примере:

|  |
| --- |
| С# |
| // Person is defined in the previous example.  //public struct Person  //{  // public string Name;  // public int Age;  // public Person(string name, int age)  // {  // Name = name;  // Age = age;  // }  //}  Person p1 = new Person("Wallace", 75);  Person p2;  p2.Name = "Wallace";  p2.Age = 75;  if (p2.Equals(p1))  Console.WriteLine("p2 and p1 have the same values.");  // Output: p2 and p1 have the same values. |

В реализации System.ValueTypeEquals используется отражение, так как необходимо определить поля, имеющиеся в любой структуре. При создании собственных структур переопределите метод Equals для предоставления эффективного алгоритма равенства, соответствующего вашему типу.

Чтобы определить, равны ли значения полей в двух экземплярах класса, можно воспользоваться методом Equals или оператором ==. Однако их следует использовать, только если они переопределены или перегружены классом с целью предоставления пользовательского определение равенства для объектов этого типа. Класс может также реализовывать интерфейс IEquatable<T> или интерфейс IEqualityComparer<T>. Оба интерфейса предоставляют методы, которые можно использовать для проверки равенства значений. При создании собственных классов, переопределяющих Equals, обязательно выполните инструкции из руководства по определению равенства значений для типа и Object.Equals(Object).

### Наследование

Наследование, вместе с инкапсуляцией и полиморфизмом, является одной из трех основных характеристик объектно-ориентированного программирования. Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют поведение, определенное в других классах. Класс, члены которого наследуются, называется базовым классом, а класс, который наследует эти члены, называется производным классом. Производный класс может иметь только один прямой базовый класс. Однако наследование является транзитивным. Если ClassC является производным от ClassB, а ClassB — от ClassA, ClassC наследует члены, объявленные в ClassB и ClassA.

|  |
| --- |
| Примечание!  Структуры не поддерживают наследование, но могут реализовывать интерфейсы. |

Концептуально производный класс является специализацией базового класса. Например, при наличии базового класса Animal возможно наличие одного производного класса, который называется Mammal, и еще одного производного класса, который называется Reptile. Mammal является Animal и Reptile является Animal, но каждый производный класс представляет разные специализации базового класса.

Объявления интерфейса могут определять реализацию по умолчанию для членов. Эти реализации наследуются производными интерфейсами и классами, реализующими эти интерфейсы.

При определении класса для наследования от другого класса производный класс явно получает все члены базового класса за исключением конструкторов и методов завершения. Производный класс повторно использует код в базовом классе без необходимости его повторной реализации. В производный класс можно добавить дополнительные члены. Производный класс расширяет функциональность базового класса.

На следующем рисунке показан класс WorkItem, представляющий рабочий элемент в бизнес-процессе. Как и другие классы, он является производным от System.Object и наследует все его методы. В WorkItem имеется пять собственных членов. К ним относится конструктор, так как конструкторы не наследуются. Класс ChangeRequest наследует от WorkItem и представляет конкретный вид рабочего элемента. ChangeRequest добавляет еще два члена к членам, унаследованным от WorkItem и Object. Он должен добавить собственный конструктор, и он также добавляет originalItemID. Свойство originalItemID позволяет ChangeRequest связать экземпляр с исходным объектом WorkItem, к которому применен запрос на изменение.

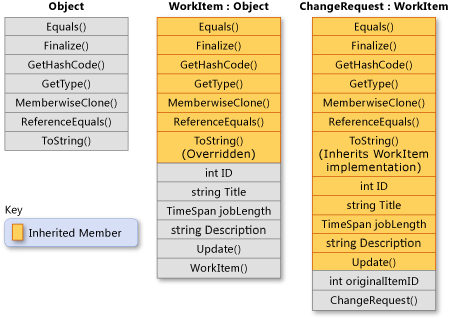


Рисунок 1. Класс WorkItem

В следующем примере показано, как выражаются в C# отношения между классами, продемонстрированные на предыдущем рисунке. В примере также показано, как WorkItem переопределяет виртуальный метод Object.ToString и как класс ChangeRequest наследует WorkItem реализацию метода. В первом блоке определяются классы:

|  |
| --- |
| С# |
| // WorkItem implicitly inherits from the Object class.  public class WorkItem  {  // Static field currentID stores the job ID of the last WorkItem that  // has been created.  private static int currentID;  //Properties.  protected int ID { get; set; }  protected string Title { get; set; }  protected string Description { get; set; }  protected TimeSpan jobLength { get; set; }  // Default constructor. If a derived class does not invoke a base-  // class constructor explicitly, the default constructor is called  // implicitly.  public WorkItem()  {  ID = 0;  Title = "Default title";  Description = "Default description.";  jobLength = new TimeSpan();  }  // Instance constructor that has three parameters.  public WorkItem(string title, string desc, TimeSpan joblen)  {  this.ID = GetNextID();  this.Title = title;  this.Description = desc;  this.jobLength = joblen;  }  // Static constructor to initialize the static member, currentID. This  // constructor is called one time, automatically, before any instance  // of WorkItem or ChangeRequest is created, or currentID is referenced.  static WorkItem() => currentID = 0;  // currentID is a static field. It is incremented each time a new  // instance of WorkItem is created.  protected int GetNextID() => ++currentID;  // Method Update enables you to update the title and job length of an  // existing WorkItem object.  public void Update(string title, TimeSpan joblen)  {  this.Title = title;  this.jobLength = joblen;  }  // Virtual method override of the ToString method that is inherited  // from System.Object.  public override string ToString() =>  $"{this.ID} - {this.Title}";  }  // ChangeRequest derives from WorkItem and adds a property (originalItemID)  // and two constructors.  public class ChangeRequest : WorkItem  {  protected int originalItemID { get; set; }  // Constructors. Because neither constructor calls a base-class  // constructor explicitly, the default constructor in the base class  // is called implicitly. The base class must contain a default  // constructor.  // Default constructor for the derived class.  public ChangeRequest() { }  // Instance constructor that has four parameters.  public ChangeRequest(string title, string desc, TimeSpan jobLen,  int originalID)  {  // The following properties and the GetNexID method are inherited  // from WorkItem.  this.ID = GetNextID();  this.Title = title;  this.Description = desc;  this.jobLength = jobLen;  // Property originalItemId is a member of ChangeRequest, but not  // of WorkItem.  this.originalItemID = originalID;  }  } |

В следующем блоке показано, как использовать базовый и производный классы:

|  |
| --- |
| С# |
| // Create an instance of WorkItem by using the constructor in the  // base class that takes three arguments.  WorkItem item = new WorkItem("Fix Bugs",  "Fix all bugs in my code branch",  new TimeSpan(3, 4, 0, 0));  // Create an instance of ChangeRequest by using the constructor in  // the derived class that takes four arguments.  ChangeRequest change = new ChangeRequest("Change Base Class Design",  "Add members to the class",  new TimeSpan(4, 0, 0),  1);  // Use the ToString method defined in WorkItem.  Console.WriteLine(item.ToString());  // Use the inherited Update method to change the title of the  // ChangeRequest object.  change.Update("Change the Design of the Base Class",  new TimeSpan(4, 0, 0));  // ChangeRequest inherits WorkItem's override of ToString.  Console.WriteLine(change.ToString());  /\* Output:  1 - Fix Bugs  2 - Change the Design of the Base Class  \*/ |

### Абстрактные и виртуальные методы

Когда базовый класс объявляет метод как virtual, производный класс может override метод с помощью своей собственной реализации. Если базовый класс объявляет член как abstract, этот метод должен быть переопределен в любом неабстрактном классе, который прямо наследует от этого класса. Если производный класс сам является абстрактным, то он наследует абстрактные члены, не реализуя их. Абстрактные и виртуальные члены являются основой для полиморфизма, который является второй основной характеристикой объектно-ориентированного программирования.

### Абстрактные базовые классы

Можно объявить класс как абстрактный, если необходимо предотвратить прямое создание экземпляров с помощью оператора new. Абстрактный класс можно использовать, только если новый класс является производным от него. Абстрактный класс может содержать один или несколько сигнатур методов, которые сами объявлены в качестве абстрактных. Эти сигнатуры задают параметры и возвращают значение, но не имеют реализации (тела метода). Абстрактному классу необязательно содержать абстрактные члены; однако если класс все же содержит абстрактный член, то сам класс должен быть объявлен в качестве абстрактного. Производные классы, которые сами не являются абстрактными, должны предоставить реализацию для любых абстрактных методов из абстрактного базового класса.

## **Полиморфизм**

Полиморфизм часто называется третьим столпом объектно-ориентированного программирования после инкапсуляции и наследования. Полиморфизм — слово греческого происхождения, означающее "многообразие форм" и имеющее несколько аспектов.

* Во время выполнения объекты производного класса могут обрабатываться как объекты базового класса в таких местах, как параметры метода и коллекции или массивы. Когда возникает полиморфизм, объявленный тип объекта перестает соответствовать своему типу во время выполнения.
* Базовые классы могут определять и реализовывать виртуальные методы, а производные классы — переопределять их, т. е. предоставлять свое собственное определение и реализацию. Во время выполнения, когда клиент вызывает метод, CLR выполняет поиск типа объекта во время выполнения и вызывает перезапись виртуального метода. В исходном коде можно вызвать метод в базовом классе и обеспечить выполнение версии метода, относящейся к производному классу.

Виртуальные методы позволяют работать с группами связанных объектов универсальным способом. Представим, например, приложение, позволяющее пользователю создавать различные виды фигур на поверхности для рисования. Во время компиляции вы еще не знаете, какие именно виды фигур создаст пользователь. При этом приложению необходимо отслеживать все различные типы создаваемых фигур и обновлять их в ответ на движения мыши. Для решения этой проблемы можно использовать полиморфизм, выполнив два основных действия.

1. Создать иерархию классов, в которой каждый отдельный класс фигур является производным из общего базового класса.
2. Применить виртуальный метод для вызова соответствующего метода на любой производный класс через единый вызов в метод базового класса.

Для начала создайте базовый класс с именем Shape и производные классы, например Rectangle, Circle и Triangle. Присвойте классу Shape виртуальный метод с именем Draw и переопределите его в каждом производном классе для рисования конкретной фигуры, которую этот класс представляет. Создайте объект List<Shape> и добавьте в него Circle, Triangleи Rectangle.

|  |
| --- |
| С# |
| public class Shape  {  // A few example members  public int X { get; private set; }  public int Y { get; private set; }  public int Height { get; set; }  public int Width { get; set; }    // Virtual method  public virtual void Draw()  {  Console.WriteLine("Performing base class drawing tasks");  }  }  public class Circle : Shape  {  public override void Draw()  {  // Code to draw a circle...  Console.WriteLine("Drawing a circle");  base.Draw();  }  }  public class Rectangle : Shape  {  public override void Draw()  {  // Code to draw a rectangle...  Console.WriteLine("Drawing a rectangle");  base.Draw();  }  }  public class Triangle : Shape  {  public override void Draw()  {  // Code to draw a triangle...  Console.WriteLine("Drawing a triangle");  base.Draw();  }  } |

Для обновления поверхности рисования используйте цикл foreach, чтобы выполнить итерацию списка и вызвать метод Draw на каждом объекте Shape в списке. Несмотря на то, что каждый объект в списке имеет объявленный тип Shape, будет вызван тип времени выполнения (переопределенная версия метода в каждом производном классе).

|  |
| --- |
| С# |
| // Polymorphism at work #1: a Rectangle, Triangle and Circle  // can all be used whereever a Shape is expected. No cast is  // required because an implicit conversion exists from a derived  // class to its base class.  var shapes = new List<Shape>  {  new Rectangle(),  new Triangle(),  new Circle()  };  // Polymorphism at work #2: the virtual method Draw is  // invoked on each of the derived classes, not the base class.  foreach (var shape in shapes)  {  shape.Draw();  }  /\* Output:  Drawing a rectangle  Performing base class drawing tasks  Drawing a triangle  Performing base class drawing tasks  Drawing a circle  Performing base class drawing tasks  \*/ |

### Обзор полиморфизма

### Виртуальные члены

Если производный класс наследуется от базового, он получает все его методы, поля, свойства и события. Конструктор производного класса может выбирать различные варианты поведения виртуальных методов:

Производный класс может переопределять виртуальные члены в базовом классе, определяя новое поведение.

Производный класс наследует ближайший метод базового класса без его переопределения, сохраняя существующее поведение, но позволяя дальнейшим производным классам переопределять метод.

Производный класс может определить новую, невиртуальную реализацию тех членов, которые скрывают реализации базового класса.

Производный класс может переопределить член базового класса, только если последний будет объявлен виртуальным или абстрактным. Производный член должен использовать ключевое слово override, указывающее, что метод предназначен для участия в виртуальном вызове. Примером является следующий код:

|  |
| --- |
| С# |
| public class BaseClass  {  public virtual void DoWork() { }  public virtual int WorkProperty  {  get { return 0; }  }  }  public class DerivedClass : BaseClass  {  public override void DoWork() { }  public override int WorkProperty  {  get { return 0; }  }  } |

Поля не могут быть виртуальными. Виртуальными могут быть только методы, свойства, события и индексаторы. Когда производный класс переопределяет виртуальный член, он вызывается даже в то случае, если доступ к экземпляру этого класса осуществляется в качестве экземпляра базового класса. Примером является следующий код:

|  |
| --- |
| С# |
| public class BaseClass  {  public virtual void DoWork() { }  public virtual int WorkProperty  {  get { return 0; }  }  }  public class DerivedClass : BaseClass  {  public override void DoWork() { }  public override int WorkProperty  {  get { return 0; }  }  } |

Виртуальные методы и свойства позволяют производным классам расширять базовый класс без необходимости использовать реализацию базового класса метода. Дополнительные сведения см. в разделе Управление версиями с помощью ключевых слов Override и New. Еще одну возможность определения метода или набора методов, реализация которых оставлена производным классам, дает интерфейс.

### Сокрытие членов базового класса новыми членами

Если вы хотите, чтобы производный класс имел член с тем же именем, что и член в базовом классе, можно использовать ключевое слово new, чтобы скрыть член базового класса. Ключевое слово new вставляется перед типом возвращаемого значения замещаемого члена класса. Примером является следующий код:

|  |
| --- |
| С# |
| public class BaseClass  {  public void DoWork() { WorkField++; }  public int WorkField;  public int WorkProperty  {  get { return 0; }  }  }  public class DerivedClass : BaseClass  {  public new void DoWork() { WorkField++; }  public new int WorkField;  public new int WorkProperty  {  get { return 0; }  }  } |

Доступ к скрытым членам базового класса можно осуществлять из клиентского кода приведением экземпляра производного класса к экземпляру базового класса. Пример:

|  |
| --- |
| С# |
| DerivedClass B = new DerivedClass();  B.DoWork(); // Calls the new method.  BaseClass A = (BaseClass)B;  A.DoWork(); // Calls the old method. |

### Защита виртуальных членов от переопределения производными классами

Виртуальные члены остаются виртуальными независимо от количества классов, объявленных между виртуальным членом и классом, который объявил его изначально. Если класс A объявляет виртуальный член, класс B является производным от класса A, а класс C — от класса B, то класс C наследует виртуальный член и может переопределить его независимо от того, объявляет ли класс B переопределение этого члена. Примером является следующий код:

|  |
| --- |
| С# |
| public class A  {  public virtual void DoWork() { }  }  public class B : A  {  public override void DoWork() { }  } |

Производный класс может остановить виртуальное наследование, объявив переопределение как запечатанное. Для остановки наследования в объявление члена класса нужно вставить ключевое слово sealed перед ключевым словом override. Примером является следующий код:

|  |
| --- |
| С# |
| public class C : B  {  public sealed override void DoWork() { }  } |

В предыдущем примере метод DoWork более не является виртуальным ни для одного класса, производного от класса C. Он по-прежнему является виртуальным для экземпляров класса C, даже если они приводятся к типу B или типу A. Запечатанные методы можно заменить производными классами с помощью ключевого слова new, как показано в следующем примере:

|  |
| --- |
| С# |
| public class D : C  {  public new void DoWork() { }  } |

В этом случае, если DoWork вызывается для D с помощью переменной типа D, вызывается новый DoWork. Если переменная типа C, B или A используется для доступа к экземпляру D, вызов DoWork будет выполняться по правилам виртуального наследования и направлять эти вызовы в реализацию DoWork в классе C.

### Доступ к виртуальным членам базового класса из производных классов

Производный класс, который заменил или переопределил метод или свойство, может получить доступ к методу или свойству на базовом классе с помощью ключевого слова base. Примером является следующий код:

|  |
| --- |
| С# |
| public class Base  {  public virtual void DoWork() {/\*...\*/ }  }  public class Derived : Base  {  public override void DoWork()  {  //Perform Derived's work here  //...  // Call DoWork on base class  base.DoWork();  }  } |

|  |
| --- |
| Примечание!  Рекомендуется, чтобы виртуальные члены использовали base для вызова реализации базового класса этого члена в их собственной реализации. Разрешение поведения базового класса позволяет производному классу концентрироваться на реализации поведения, характерного для производного класса. Если реализация базового класса не вызывается, производный класс сопоставляет свое поведение с поведением базового класса по своему усмотрению. |

## **Отношения между классами**

Мы можем выделить несколько основных отношений: наследование, реализация, ассоциация, композиция и агрегация.

### Наследование

Наследование является базовым принципом ООП и позволяет одному классу (наследнику) унаследовать функционал другого класса (родительского). Нередко отношения наследования еще называют генерализацией или обобщением. Наследование определяет отношение IS A, то есть "является". Например:

|  |
| --- |
| С# |
| class User  {      public int Id { get; set; }      public string Name { get; set; }  }    class Manager : User  {      public string Company{ get; set; }  } |

В данном случае используется наследование, а объекты класса Manager также являются и объектами класса User.

С помощью диаграмм UML отношение между классами выражается в незакрашенной стрелочке от класса-наследника к классу-родителю:

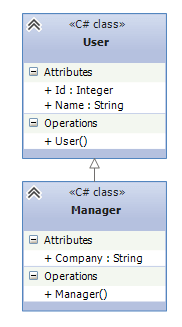


Рисунок 2. Связи наследования

### Реализация

Реализация предполагает определение интерфейса и его реализация в классах. Например, имеется интерфейс IMovable с методом Move, который реализуется в классе Car:

|  |
| --- |
| С# |
| public interface IMovable  {      void Move();  }  public class Car : IMovable  {      public void Move()      {          Console.WriteLine("Машина едет");      }  } |

С помощью диаграмм UML отношение реализации также выражается в незакрашенной стрелочке от класса к интерфейсу, только линия теперь пунктирная:

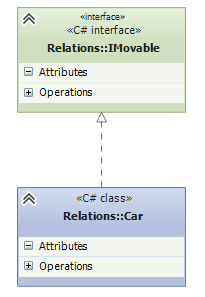


Рисунок 3. Связи реализации

### Ассоциация

Ассоциация — это отношение, при котором объекты одного типа неким образом связаны с объектами другого типа. Например, объект одного типа содержит или использует объект другого типа. Например, игрок играет в определенной команде:

|  |
| --- |
| С# |
| class Team  {    }  class Player  {      public Team Team { get; set; }  } |

Класс Player связан отношением ассоциации с клаcсом Team. На схемах UML ассоциация обозначается в виде обычно стрелки:

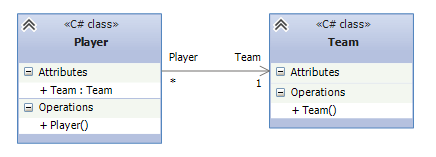


Рисунок 4. Связи ассоциации

Нередко при отношении ассоциации указывается кратность связей. В данном случае единица у Team и звездочка у Player на диаграмме отражает связь 1 ко многим. То есть одна команда будет соответствовать многим игрокам.

Агрегация и композиция являются частными случаями ассоциации.

### Композиция

Композиция определяет отношение HAS A, то есть отношение "имеет". Например, в класс автомобиля содержит объект класса электрического двигателя:

|  |
| --- |
| С# |
| public class ElectricEngine  { }    public class Car  {      ElectricEngine engine;      public Car()      {          engine = new ElectricEngine();      }  } |

При этом класс автомобиля полностью управляет жизненным циклом объекта двигателя. При уничтожении объекта автомобиля в области памяти вместе с ним будет уничтожен и объект двигателя. И в этом плане объект автомобиля является главным, а объект двигателя - зависимой.

На диаграммах UML отношение композиции проявляется в обычной стрелке от главной сущности к зависимой, при этом со стороны главной сущности, которая содержит, объект второй сущности, располагается закрашенный ромбик:

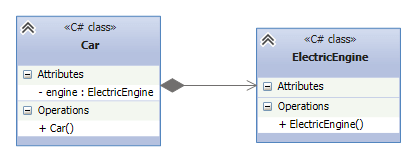


Рисунок 5. Связи композиции

### Агрегация

От композиции следует отличать агрегацию. Она также предполагает отношение HAS A, но реализуется она иначе:

|  |
| --- |
| С# |
| public abstract class Engine  { }    public class Car  {      Engine engine;      public Car(Engine eng)      {          engine = eng;      }  } |

При агрегации реализуется слабая связь, то есть в данном случае объекты Car и Engine будут равноправны. В конструктор Car передается ссылка на уже имеющийся объект Engine. И, как правило, определяется ссылка не на конкретный класс, а на абстрактный класс или интерфейс, что увеличивает гибкость программы.

Отношение агрегации на диаграммах UML отображается также, как и отношение композиции, только теперь ромбик будет незакрашенным:

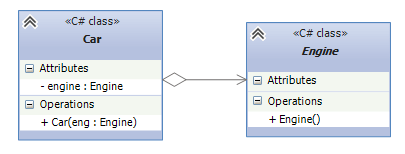


Рисунок 6. Связи Агрегация

При проектировании отношений между классами надо учитывать некоторые общие рекомендации. В частности, вместо наследования следует предпочитать композицию. При наследовании весь функционал класса-наследника жестко определен на этапе компиляции. И во время выполнения программы мы не можем его динамически переопределить. А класс-наследник не всегда может переопределить код, который определен в родительском классе. Композиция же позволяет динамически определять поведение объекта во время выполнения, и поэтому является более гибкой.

Вместо композиции следует предпочитать агрегацию, как более гибкий способ связи компонентов. В то же время не всегда агрегация уместна. Например, у нас есть класс человека, который содержит объект нервной системы. Понятно, что в реальности, по крайней мере на текущий момент, невозможно вовне определить нервную систему и внедрить ее в человека. То есть в данном случае человек будет главным компонентом, а нервная система - зависимым, подчиненным, и их создание и жизненный цикл будет происходить совместно, поэтому здесь лучше выбрать композицию.

# ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Решите задание используя основные отношения
2. Спроектируйте блок схему к программе
3. Отобразить в отчете проделанные шаги
4. Ответить на контрольные вопросы
5. Написать вывод о проделанной работе

# ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

|  |
| --- |
| Вариант 1 – Определить класс Children, который содержит такие поля: закрытые – имя ребенка, фамилию и возраст, публичные – методы ввода данных и отображения их на экран. Объявить два объекта класса, внести данные и показать их. |
| Вариант 2 – Опишите, используя Класс, каталог книг в библиотеке. Составьте программу, выдающую список книг автора А.Дюма,хранящихся в библиотеке. |
| Вариант 3 – Опишите, используя Класс, записную книжку. Составьте программу, выдающую список друзей кому в этом году исполняется 20лет. Фамилия и инициалы, год рождения, день рождения и месяц рождения. |
| Вариант 4 - Опишите, используя Класс, школьную нагрузку. Фамилия преподавателя, класс, чассы.Определите у какого преподавателя самая большая нагрузка. |
| Вариант 5 – Опишите, используя Класс, вступительные экзамены. Абитуриенты сдавали три экзамена, для поступления необходимо набрать 12 баллов. Определите списки абитуриентов зачисленных в институт, количество не сдавших экзамен, список абитуриентов сдавших три экзамена на 5. |
| Вариант 6 - Опишите используя Класс, данные на учеников (фамилия, улица, дом, квартира). Составьте программу, определяющую сколько учеников живет на улице Свердлова, списки учеников, живущих в доме номер 45 |
| Вариант 7 –Опишите используя Класс, товар(наименование товара, старая цена, новая цена).Составьте программу, определяющую на какие товары повысятся ценны и на сколько процентов. |
| Вариант 8 –Дана анкета: фамилия, пол, число, месяц, год рождения. Выберите самого старшего мужчину, напечатайте все фамилии, начинающиеся с буквы “Б” и даты рождения этих людей |
| Вариант 9 –В анкетных данных обозначены фамилия, пол, рост. Определите средний рост женщин, фамилию самого высокого мужчины. |
| Вариант 10 –В экзаменационной ведомости можно выделить сведения о ведомости (предмет, номер группы, дата экзамена), сведения о человеке (фамилия, номер зачетной книжки, оценка за экзамен). Определите сколько человек не сдали информатику и математику, выдать их списки: фамилия, номер группы. |

# КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как устроена работа Реализации?
2. Как устроена работа Ассоциации?
3. Как устроена работа Агрегации?

# РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. Официальная документация Microsoft C# — Текст: электронный // Microsoft [сайт]. — URL: https://docs.microsoft.com/ (дата обращения: 12.03.2020).
2. Сообщество IT-специалистов — Текст: электронный // Habr [сайт]. — URL: https://habr.com/ (дата обращения: 12.03.2020).
3. Сайт о программирование / — Текст: электронный // Metanit [сайт]. — https://metanit.com/ (дата обращения: 12.03.2020).
4. С# Tutotial — Текст: электронный // Tutotial [сайт]. — URL: https://csharp.net-tutorials.com/ (дата обращения: 13.03.2020).

# ПРИЛОЖЕНИЕ А

**Требования к отчету**

Общие требования:

1. Шрифт – Times New Roman, 14 пт.
2. Интервалы: междустрочный – 1,5 строки, интервал до и после абзаца – 0 пт.
3. Отступ первой строки – 1,25
4. Рисунки и подписи к ним выравниваются по центру.

Требования к структуре отчета:

1. Титульный лист
2. Содержание
3. Основная часть:
   1. Задачи
   2. Код
   3. Блок-схемы
   4. Результат работы
4. Выводы по работе
5. Ответы на контрольные вопросы

# ПРИЛОЖЕНИЕ Б

ОГБПОУ «ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Отчет по учебной-практической работе №3

учебной практике   
*УП 05. Программирование*

*Тема: «Отношения между классами»*

Выполнил:

студент \_\_\_\_\_\_\_ группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:  
преподаватель

Сидиков И.Д.

г. Томск – 2020 г