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. 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 **`Shape[]`** и добавьте в него **`Circle`**, **`TriangleИ Rectangle`**.

Полиморфизм

```
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();
    }
}
```

Полиморфизм

```
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).

```
// 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 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
*/
```

Полиморфизм

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:

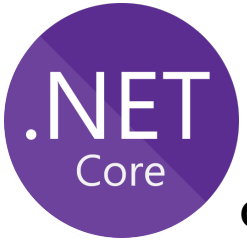
```
public class BaseClass
{
    public virtual void DoWork() { }
    public virtual int WorkProperty
    {
        get { return 0; }
    }
}
public class DerivedClass : BaseClass
{
    public override void DoWork() { }
    public override int WorkProperty
    {
        get { return 0; }
    }
}
```

Полиморфизм

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:

```
public class BaseClass
{
    public virtual void DoWork() { }
    public virtual int WorkProperty
    {
        get { return 0; }
    }
    public virtual int Property { get; set; }
    //Error
    //public virtual int Field;
}
public class DerivedClass : BaseClass
{
    public override void DoWork() { }
    public override int WorkProperty
    {
        get { return 0; }
    }
}
```



Полиморфизм

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:

```
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 double WorkField;
    public new string WorkProperty
    {
        get { return "s"; }
    }
}
```

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

```
DerivedClass B = new DerivedClass();
B.DoWork();  // Calls the new method.

BaseClass A = (BaseClass)B;
A.DoWork();  // Calls the old method.
```

Полиморфизм

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:

```
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:

```
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:

```
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**.

Полиморфизм

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:

```
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();
    }
}
```