Практическая работа 4: Реализация наследования и виртуальных методов

Цель:

- **Закрепить** понимание принципов наследования в ООП.
- **Научиться** создавать иерархии классов.
- **Освоить** механизм переопределения методов с помощью ключевых слов `virtual` и `override`.
- **Понять** полиморфизм на практике: работу с объектами производных классов через ссылки на базовый класс.

Теоретическая часть:

Наследование — это механизм ООП, позволяющий создать новый класс (производный) на основе существующего (базового). Производный класс наследует поля, свойства и методы базового класса и может добавлять свои собственные или изменять унаследованные.

Виртуальные методы — это методы базового класса, помеченные ключевым словом `virtual`. Они позволяют производному классу предоставить свою собственную реализацию этого метода, используя ключевое слово `override`. Это основа для реализации **полиморфизма**.

Полиморфизм — это возможность работать с объектами разных классов через единый интерфейс базового класса. Во время выполнения программы будет вызываться метод, соответствующий фактическому типу объекта, а не типу ссылки.

Пример:

```
class Animal // Базовый класс
{
    public virtual void MakeSound() // Виртуальный метод
    {
        Console.WriteLine("Some generic animal sound");
    }
}

class Dog : Animal // Производный класс
{
    public override void MakeSound() // Переопределение метода
    {
        Console.WriteLine("Woof! Woof!");
    }
}

// Использование
Animal myAnimal = new Dog(); // Полиморфизм
myAnimal.MakeSound(); // Выведет "Woof! Woof!", а не "Some generic animal sound"
```

Ход работы:

1. Создайте новое консольное приложение .NET с именем `InheritanceLab`.
2. Спроектируйте иерархию классов согласно вашему варианту.
3. В базовом классе создайте виртуальный метод (например, `DisplayInfo()`, `CalculateSomething()`, `MakeSound()`).
4. В производных классах переопределите этот метод, чтобы он выводил или вычислял информацию, специфичную для данного класса.
5. В методе `Main` создайте массив или список объектов базового типа, но поместите в него экземпляры различных производных классов.
6. Организуйте цикл, который вызовет переопределенный метод для каждого элемента коллекции. Продемонстрируйте работу полиморфизма.
7. Оформите код в соответствии с соглашениями C# (правильные имена классов, методов, использование модификаторов доступа).
8. Сделайте скриншоты работы вашей программы и сохраните их в папку `images`.
9. Создайте `README.md` файл с описанием работы.
10. Инициализируйте локальный git-репозиторий, создайте необходимые файлы (`.gitignore`), закоммитьте изменения и запушите проект в ваш локальный Gogs.

Практический пример (Образец выполнения для варианта 0):

Задание: Создайте иерархию классов "Фигура". Базовый класс `Shape` с виртуальным методом `Draw()`. Производные классы: `Circle`, `Rectangle`, `Triangle`. Переопределите метод `Draw()` так, чтобы он выводил в консоль название фигуры.

Код:

`Shape.cs`

```
namespace InheritanceLab
{
    public class Shape
    {
        public virtual void Draw()
        {
            Console.WriteLine("Drawing a generic shape");
        }
    }
}
```

`Circle.cs`

```
namespace InheritanceLab
{
    public class Circle : Shape
    {
        public override void Draw()
```

```
        {
            Console.WriteLine("Drawing a Circle");
            // Можно добавить ASCII-арт :)
            // Console.WriteLine("   ***   ");
            // Console.WriteLine(" *     * ");
            // Console.WriteLine("*       *");
            // Console.WriteLine(" *     * ");
            // Console.WriteLine("   ***   ");
        }
    }
}
```

Rectangle.cs

```
namespace InheritanceLab
{
    public class Rectangle : Shape
    {
        public override void Draw()
        {
            Console.WriteLine("Drawing a Rectangle");
        }
    }
}
```

Program.cs

```
namespace InheritanceLab
{
    class Program
    {
        static void Main(string[] args)
        {
            // Создаем список фигур (полиморфная коллекция)
            List<Shape> shapes = new List<Shape>();

            // Добавляем объекты разных типов, но все они - Shape
            shapes.Add(new Circle());
            shapes.Add(new Rectangle());
            shapes.Add(new Triangle());
            shapes.Add(new Circle());

            Console.WriteLine("Drawing all shapes:");
            Console.WriteLine("--------------------");

            // Полиморфизм в действии!
            // Для каждого объекта в списке вызывается его собственный метод
Draw()
            foreach (Shape shape in shapes)
```

```
            {
                shape.Draw();
            }

            Console.ReadLine();
        }
    }
}
```

Вывод в консоль:

```
Drawing all shapes:
-------------------
Drawing a Circle
Drawing a Rectangle
Drawing a Triangle
Drawing a Circle
```

Варианты заданий (для студентов):

Студент выбирает вариант, соответствующий его номеру в журнале.

1. **Транспортные средства:** `Vehicle` (базовый), `Car`, `Bicycle`, `Motorcycle`. Виртуальный метод `StartEngine()` (для велосипеда может выводить "У велосипеда нет двигателя").
2. **Сотрудники:** `Employee` (базовый), `Manager`, `Developer`, `Intern`. Виртуальный метод `CalculateBonus()` (рассчитывает бонус в зависимости от должности).
3. **Банковские счета:** `Account` (базовый), `SavingsAccount`, `CheckingAccount`, `CreditAccount`. Виртуальный метод `CalculateInterest()` (начисляет процент по-разному для разных типов счетов).
4. **Животные:** `Animal` (базовый), `Mammal`, `Bird`, `Reptile`. Виртуальный метод `MakeSound()`.
5. **Геометрические фигуры (с расчетами):** `Shape` (базовый), `Square`, `Circle`, `Rectangle`. Виртуальный метод `CalculateArea()`.
6. **Устройства ввода:** `InputDevice` (базовый), `Keyboard`, `Mouse`, `Scanner`. Виртуальный метод `ProcessInput()`.
7. **Учебные заведения:** `EducationalInstitution` (базовый), `School`, `University`, `College`. Виртуальный метод `DisplayDescription()`.
8. **Уведомления:** `Notification` (базовый), `EmailNotification`, `SMSNotification`, `PushNotification`. Виртуальный метод `Send()`.
9. **Компьютерные компоненты:** `ComputerPart` (базовый), `CPU`, `GPU`, `RAM`. Виртуальный метод `DisplaySpecs()`.
10. **Еда и напитки:** `Product` (базовый), `Fruit`, `Vegetable`, `Beverage`. Виртуальный метод `GetExpiryInfo()`.
11. **Документы:** `Document` (базовый), `Invoice`, `Report`, `Contract`. Виртуальный метод `Print()`.
12. **Музыкальные инструменты:** `MusicalInstrument` (базовый), `Guitar`, `Piano`, `Drums`. Виртуальный метод `Play()`.

13. **Игровые персонажи:** `Character` (базовый), `Warrior`, `Mage`, `Archer`. Виртуальный метод `UseSpecialAbility()`.
14. **Библиотечные ресурсы:** `LibraryItem` (базовый), `Book`, `Journal`, `DVD`. Виртуальный метод `CheckOut()`.
15. **Погодные явления:** `WeatherPhenomenon` (базовый), `Rain`, `Snow`, `Fog`. Виртуальный метод `Describe()`.

Критерии оценки:

- **5 (Отлично):**
  - Программа компилируется и выполняется без ошибок.
  - Полностью и корректно реализована иерархия классов согласно варианту.
  - Виртуальный метод в базовом классе и его переопределение в производных классах реализованы верно.
  - Ярко продемонстрирован полиморфизм (работа с коллекцией базового типа).
  - Код чистый, хорошо отформатирован, используются понятные имена.
  - Все файлы репозитория (`README.md`, `.gitignore`, скриншоты) созданы правильно.
- **4 (Хорошо):**
  - Программа работает, но есть незначительные недочеты (например, неидеальное форматирование кода).
  - Реализована иерархия и полиморфизм, но в логике методов есть небольшие погрешности.
  - `README.md` или скриншоты оформлены не полностью.
- **3 (Удовлетворительно):**
  - Программа в основном работает, но с ошибками при выполнении (например, не для всех объектов вызывается правильный метод).
  - Иерархия классов создана, но механизм `virtual`/`override` применен не для всех необходимых методов.
  - Отсутствуют некоторые элементы репозитория (скриншоты, `.gitignore`).
- **2 (Неудовлетворительно):**
  - Программа не компилируется или не работает.
  - Иерархия классов не построена.
  - Механизм виртуальных методов не используется.
  - Репозиторий не оформлен.

Контрольные вопросы:

1. Что такое наследование и какова его основная цель?
2. В чем разница между ключевыми словами `virtual` и `override`?
3. Что такое полиморфизм и как он связан с виртуальными методами?
4. Можно ли переопределить невиртуальный метод? Если нет, то почему?
5. Что произойдет, если в производном классе не переопределить виртуальный метод?
6. Каков порядок вызова конструкторов при создании объекта производного класса?
7. Объясните, что такое "сокрытие метода" (method hiding) с помощью `new` и чем оно отличается от переопределения.
8. Можно ли объявить виртуальным статический метод? Объясните свой ответ.

Структура репозитория для Gogs

Ваш репозиторий на Gogs должен иметь следующую структуру:

```
InheritanceLab/
├── InheritanceLab.sln
├── InheritanceLab/
│   ├── Program.cs
│   ├── [Файлы базового и производных классов].cs
│   └── bin/, obj/ (игнорируются через .gitignore)
├── images/
│   ├── screenshot1.png
│   └── screenshot2.png
├── .gitignore
└── README.md
```

Содержимое файла `.gitignore`

Используйте стандартный `.gitignore` для C#. Его можно взять, например, с GitHub: https://github.com/github/gitignore/blob/main/VisualStudio.gitignore

Пример содержимого файла `README.md`

```
# Практическая работа №X: Реализация наследования и виртуальных методов

**Вариант: 1**
**Студент: Иванов И.И.**

## Задание
Создайте иерархию классов "Транспортные средства". Базовый класс `Vehicle` должен
содержать виртуальный метод `StartEngine()`. Создайте производные классы: `Car`,
`Bicycle`, `Motorcycle`. В каждом производном классе переопределите метод
`StartEngine()`, чтобы он выводил соответствующее сообщение:
- Car: "Двигатель автомобиля запущен."
- Bicycle: "У велосипеда нет двигателя."
- Motorcycle: "Мотоцикл завелся с рёвом."

Продемонстрируйте работу полиморфизма, создав коллекцию объектов базового типа
`Vehicle` и вызвав для каждого из них метод `StartEngine()`.

## Код программы

### Базовый класс Vehicle (Vehicle.cs)

public class Vehicle
{
    public virtual void StartEngine()
    {
```

```
        Console.WriteLine("Двигатель транспортного средства запущен.");
    }
}
```

Производный класс Car (Car.cs)

```
public class Car : Vehicle
{
    public override void StartEngine()
    {
        Console.WriteLine("Двигатель автомобиля запущен.");
    }
}
```

(Аналогично для Bicycle и Motorcycle)

Основной код программы (Program.cs)

```
class Program
{
    static void Main(string[] args)
    {
        List<Vehicle> vehicles = new List<Vehicle>
        {
            new Car(),
            new Bicycle(),
            new Motorcycle(),
            new Car()
        };

        Console.WriteLine("Запуск двигателей:");
        foreach (Vehicle v in vehicles)
        {
            v.StartEngine();
        }
    }
}
```