Тема 10. Наследование

Парадигмы программирования. C#

План

- Наследование классов
- Реализация методов
- Запечатанные классы
- Использование интерфейсов
- Использование абстрактных классов

Наследование классов

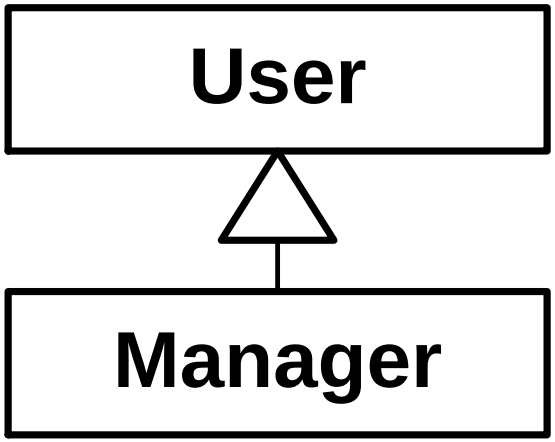
Часть 1

Наследование конкретных классов

- Синтаксис

```
class User
{
}

class Manager : User
{
}
```



- Наследуемый класс получает от базового класса большинство его элементов
- Наследуемый класс не может быть более доступным по сравнению с базовым классом

Обращение к членам базового класса

- Наследуемые защищённые члены (имеющие модификатор доступа `protected`) неявно являются защищёнными и в классе-наследнике
- Методы класса-наследника могут получать доступ только к наследуемым защищённым членам своего класса-предка
- Модификатор доступа `protected` не может применяться в структурах

Пример обращения к членам базового класса

```
class User
{
    protected string name;
}

class Manager : User
{
    public string GetName()
    {
        return name;
    }
}
class Program
{
    public void Try(User user)
    {
        user.name = "Vasya"; // Ошибка компиляции
    }
}
```

Вызов конструктора базового класса

- Определения конструкторов для вызова конструктора базового класса должны использовать ключевое слово `base`

```
class User
{
    protected User(string name) { }
}

class Manager : User
{
    public Manager(string name) : base(name) { }
}
```

Инициализатор конструктора

- Если конструктор класса-наследника явно не вызывает конструктор класса-предка, то компилятор неявно добавляет инициализатор конструктора базового класса
- К частному конструктору (имеющему модификатор доступа private) нельзя обращаться из класса-наследника
- Для разрешения столкновения областей видимости используйте ключевое слово `base`

Реализация методов

Часть 2

Определение виртуальных методов

- Виртуальные методы являются полиморфическими
- Виртуальные методы обязаны быть не пустыми

```
class User
{
    public virtual string GetName()
    {
    ...
    }

    public virtual void SetName(
        string name); // Ошибка компиляции
}
```

Ограничения виртуальных методов

- Для использования виртуальных методов
  - Недопустимо определять статические виртуальные методы
  - Недопустимо определять частные виртуальные методы

Переопределение виртуальных методов

- Для переопределения виртуальных методов в классах-наследниках необходимо использовать ключевое слово `override`
- Переопределяемый метод не может быть пустым, поскольку он тоже является виртуальным методом

```
class User
{
    protected string name;
    public virtual string GetName()
    {
        return name;
    }
}

class Manager : User
{
    public override string GetName()
    {
        return String.Format(
            "Hello, {0}!", base.GetName());
    }
}
```

Работа с переопределяемыми методами

- Допустимо переопределять только идентичные виртуальные методы

```
class User
{
    public virtual string GetName() { ... }
    public void SetName(string name) { ... }
}

class Manager : User
{
    public override string GetName() { ... }
    // Ошибка компиляции: аналогичный метод в
    // классе-предке не является виртуальным
    public override void SetName(string name) { ... }
}
```

- Необходимо добиваться совпадения у переопределяемого метода и виртуального метода
  - Модификатора доступа
  - Возвращаемого типа данных
  - Идентификатора
  - Параметров
- Допустимо переопределять переопределённый метод
- Недопустимо явно определять переопределённый метод, как виртуальный
- Переопределённый метод не может быть статическим или частным

Сокрытие методов

- Для сокрытия метода базового класса используйте ключевое слово `new`

```
class User
{
    public void SetName(string name)
    {
    }
}

class Manager : User
{
    public new void SetName(string name)
    {
    }
}
```

Использование ключевого слова new

- Сокрытие и виртуальных, и не виртуальных методов
- Разрешение столкновений имён в коде
- Сокрытие методов, имеющих идентичные сигнатуры

Что будет напечатано на экране?

```
class A
{
    public virtual void M() { Console.Write("A"); }
}
class B : A
{
    public override void M() { Console.Write("B"); }
}
class C : B
{
    new public virtual void M() { Console.Write("C"); }
}
class D : C
{
    public override void M() { Console.Write("D"); }

    static void Main()
    {
        D d = new D(); C c = d; B b = c; A a = b;
        d.M(); c.M(); b.M(); a.M();
    }
}
```

Найдите ошибки

```
class Base
{
    public void Alpha() { ... }
    public virtual void Beta() { ... }
    public virtual void Gamma(int i) { ... }
    public virtual void Delta() { ... }
    private virtual void Epsilon() { ... }
}
class Derived : Base
{
    public override void Alpha() { ... }
    protected override void Beta() { ... }
    public override void Gamma(double d) { ... }
    public override int Delta() { ... }
}
```

Запечатанные классы

Часть 3

Использование запечатанных классов

- От запечатанных классов нельзя наследоваться
- Допустимо использовать запечатанные классы для оптимизации операций во время выполнения программы
  - В запечатанном классе компилятор может заменить полиморфические методы на обычные
- Множество классов в .NET Framework являются запечатанными: String, Array, StringBuilder и т.д.
- Для создания запечатанных классов используёте ключевое слово sealed

Пример использования запечатанного класса

```
sealed class User
{
}
// Ошибка компиляции:
// нельзя наследоваться
// от запечатанных классов
class Manager : User
{
}
```

Использование интерфейсов

Часть 4

Определение интерфейсов

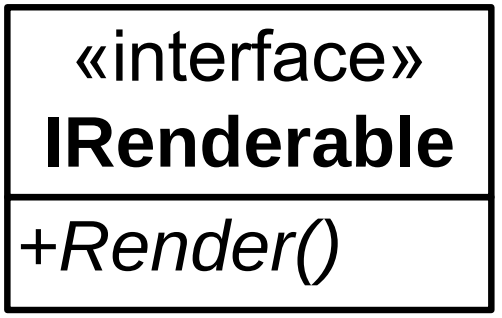
- Для обозначения интерфейса используется ключевое слово interface

Идентификатор интерфейса начинается с «I»

```
interface IRenderable
{
    void Render();
}
```

Нет модификаторов доступа

Нет тел методов

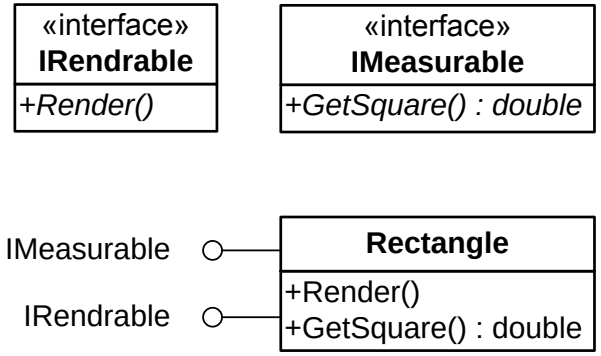


Реализация нескольких интерфейсов

- Класс может реализовывать один или несколько интерфейсов

```
interface IRenderable
{
    void Render();
}
interface IMeasurable
{
    double GetSquare();
}
class Rectangle : IRenderable, IMeasurable
{
    public void Render() { }

    public double GetSquare() { }
}
```



- Интерфейс может расширять ноль или несколько интерфейсов
- Класс может быть более доступным, чем его базовые интерфейсы
- Интерфейс не может быть более доступным, чем его базовые интерфейсы
- Класс должен реализовать все наследуемые методы интерфейса

Реализация методов интерфейса

- Метод, реализующий метод интерфейса, должен быть таким же, как и метод интерфейса
  - Модификатор доступа должен совпадать (public)
  - Должен совпадать тип возвращаемого значения
  - Должен совпадать идентификатор
  - Должны совпадать параметры
- Метод, реализующий метод интерфейса, может быть виртуальным или не виртуальным

Явная реализация методов интерфейса

- Используйте полные имена методов (включая имена интерфейсов)

```
interface IRenderable
{
    void Render();
}
interface IMeasurable
{
    double GetSquare();
}
class Rectangle : IRenderable, IMeasurable
{
    void IRenderable.Render() { }

    double IMeasurable.GetSquare() { }
}
...
Rectangle r = new Rectangle();
((IRenderable)r).Render();
```

- Ограничения явной реализации методов интерфейса
  - Допустимо получить доступ к методам только через интерфейс при вызове
  - Методы нельзя определять, как виртуальные
  - Нельзя указывать модификатор доступа

Использование абстрактных классов

Часть 5

Определение абстрактных классов

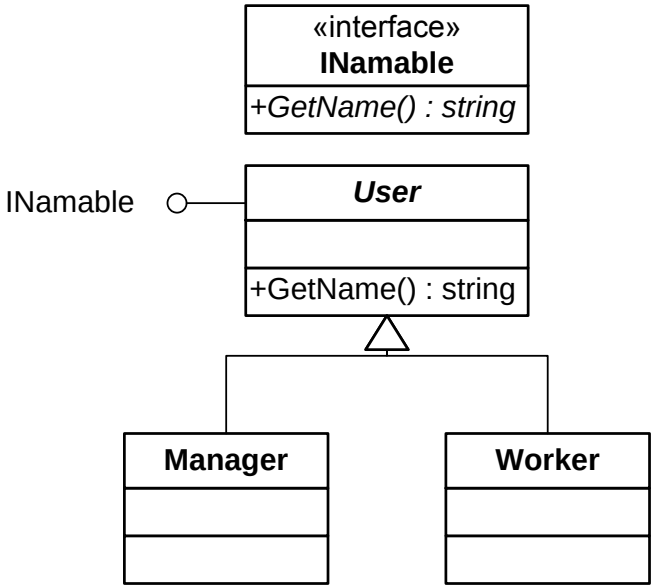
- Используйте ключевое слово `abstract`
- Внутри абстрактного класса могут располагаться абстрактные методы
- От абстрактного класса нельзя получать экземпляр

```
abstract class User
{
}

class Program
{
    static void Main()
    {
        // Ошибка компиляции:
        // нельзя создавать экземпляр
        // абстрактного класса
        User user = new User();
    }
}
```

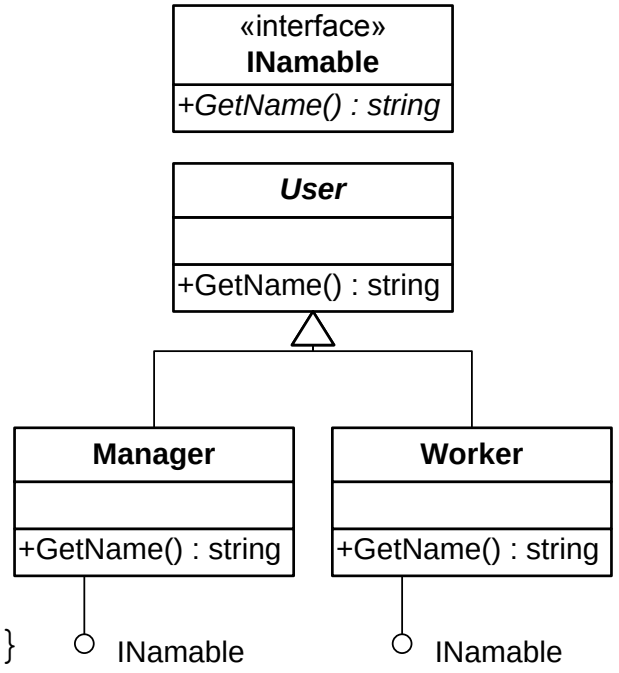
Абстрактные классы в иерархии классов

```
interface INamable
{
    string GetName();
}
abstract class User : INamable
{
    string INamable.GetName() { }
}
class Manager : User
{
}
class Worker : User
{
}
```



Абстрактные классы в иерархии классов (продолжение)

```
interface INamable
{
    string GetName();
}
abstract class User
{
    public virtual string GetName() { }
}
class Manager : User, Inamable
{
    public override string GetName() { }
}
class Worker : User, INamable
{
    public override string GetName() { }
}
```



Сравнение абстрактных классов и интерфейсов

- Сходства
  - Невозможно получить экземпляр
  - Невозможно сделать запечатанными (sealed)
- Различия
  - Интерфейсы не могут содержать никакую реализацию
  - Интерфейсы не могут содержать не публичные члены
  - Интерфейсы могут наследоваться только от интерфейсов

Реализация абстрактных методов

- Используйте ключевое слово abstract

```
abstract class User
{
    public virtual string GetName() { }
    public abstract int GetLength();
}
class Manager : User
{
    public override string GetName() { }
    public override int GetLength() { }
}
```

- Только абстрактные классы могут определять абстрактные методы
- Абстрактные методы не могут содержать тело метода

Работа с абстрактными методами

- Абстрактные методы являются виртуальными
- Переопределяемые методы могут переопределять абстрактные методы в любых классах, наследуемых от абстрактного класса
- Абстрактные методы могут переопределять методы базового класса, объявленные виртуальными
- Абстрактные методы могут переопределять методы базового класса, объявленные переопределяемыми

Заключение

- Спасибо за внимание!